uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Electrónica Industrial y Automática.
Curso académico 2018-2019

Trabajo Fin de Grado

"Implementación de función física inclonable (PUF) en FPGA"

Borja Verdasco Ayala

Tutor

Honorio Martín González

Leganés (Madrid), 11 de Julio de 2019

Resumen

Dentro del ámbito de la seguridad hardware y la ciberseguridad toman cada vez más importancia técnicas de criptografía y métodos de cifrado cuyo objetivo es garantizar la confidencialidad de la información.

La piratería dentro del sector de los circuitos integrados genera pérdidas considerables en la industria, por lo cual resulta necesaria la inversión y el desarrollo de técnicas de seguridad más fiables y vigorosas.

De esta necesidad surge la aparición de lo que se conoce como función física inclonable (PUF). Las funciones físicas inclonables se basan en la variabilidad de ciertas propiedades físicas durante el proceso de fabricación de semiconductores, generando firmas digitales para cada dispositivo imposibles de ser replicadas.

En este trabajo, la idea es esclarecer la posibilidad de explotar las variaciones en los sensores "on-chip" de distintas FPGA a través del conversor analógico-digital (XADC) integrado en las mismas, e implementando una función física inclonable (PUF) basada en "Ring Oscillators".

Palabras clave: Función Física Inclonable, Ring Oscillator, XADC, Macro, FPGA.

Abstract

Within hardware security and cybersecurity environment cryptography techniques and encryption methods become relevant. The main goal is to assure data privacy.

Concerning Integrated Circuits, industrial piracy generates economic losses affecting all sectors of Industry. Therefore, investment and development of security methods turns into a must in order to gain in reliability and robustness.

From this need arose the emergence of 'Physically Unclonable Functions' (PUFs). Indeed, Physical Unclonable Functions are based on the random variations on chip level of physical properties during the silicon manufacturing process of semiconductors. This leads to the generation of inherent "digital signatures" for each device that cannot be replicated.

This work will clarify the possibility of exploiting on-chip sensors variations of different FPGA through the Xilinx Analog to Digital Converter (XADC) which is integrated on them. And also, implementing such a PUF based on Ring Oscillators.

Dedicatoria y agradecimientos.

Deseo agradecer a mi tutor Honorio Martín toda la orientación recibida durante la elaboración de este trabajo. Por otra parte, me gustaría expresar mi total gratitud tanto a mis padres como a mis amigos sin cuyo apoyo esto no habría sido posible.

Índice.

Índice de figuras

Índice de tablas

Listado de abreviaturas y acrónimos

FPGA	Field Programmable Gate Array.
PUF	Physical Unclonable Function.
RNG	Random Number Generator.
PKC	Public Key Cryptography.
IoT	Internet of Things.
CMOS	Complementary Metal-Oxide Semiconductor.
CRP	Challenge-Response Pair.
RO(s)	Ring Oscillator(s)
VHDL	VHSIC Hardware Description Language.
SRAM	Static Random Access Memory.
ROPUF	Ring Oscillator based PUF.
SRAM-PUF	Static Random Access Memory based PUF.
IP	Intellectual Property.
XADC	Xilinx Analog to Digital Converter.
T	Temperature.
VCCINT	Main Power Supply for the FPGA's internal logic.
VCCAUX	Auxiliary power source to optimize FPGA functions.
VCCBRAM	Voltage supply for BRAM memories inside FPGA.
XDC	Xilinx Design Constraints.
RPM(s)	Relatively Placed Macro(s).
LUT	Look Up Table

1. Introducción.

Los circuitos integrados (ICs) se definen como un conjunto de componentes electrónicos, ensamblados como una única unidad, en el que dispositivos activos (como transistores o diodos), dispositivos pasivos (como condensadores y resistencias) y sus interconexiones se construyen bajo el mismo material semiconductor (normalmente Silicio) usado como sustrato. El circuito resultante es un chip monolítico de dimensiones reducidas.

La relevancia de los circuitos integrados desde su origen se ha incrementado de manera casi exponencial hasta la actualidad. En su momento, fue la base de desarrollo tecnológico durante el programa espacial y militar durante finales del siglo XX. Fruto de estos avances, se estableció una tendencia conocida como Ley de Moore la cual establece que cada dos años se duplica el número de transistores que se pueden contener en un IC del mismo tamaño. [1]

El papel fundamental que han jugado los circuitos integrados ha dado como resultado un gran desarrollo en los procesos de fabricación de tecnologías submicrónicas. En la actualidad, el uso de ICs se expande hacia diversos campos de operación como la informática, el sector de las comunicaciones e incluso la industria del transporte. Dentro del campo de la tecnología electrónica, los dispositivos FPGA han alcanzado una gran popularidad dado su capacidad de realizar distintas tareas de forma simultánea y la posibilidad de ser reprogramadas tantas veces como sea necesario.

Dentro del ámbito de los circuitos integrados, surge un problema cuyo abordaje es esencial y se trata de la piratería industrial. El concepto de la piratería industrial se basa en la falsificación de dispositivos de manera que se confunde al consumidor y se usurpa la legitimidad de las marcas registradas. [2]

Esto supone una amenaza económica desde el punto de vista de las empresas dado que no se obtienen beneficios de las ventas de esos dispositivos falsificados. Por otra parte, las imitaciones de estos productos no cumplen en su mayoría con las especificaciones técnicas y garantías que son respaldadas por los fabricantes. De esta manera, se compromete la satisfacción tanto de clientes particulares como de otras empresas a las que los fabricantes de dispositivos dan soporte. [3]

Por ende, la seguridad de estos dispositivos, entendiendo el concepto de seguridad basada en la autenticación de dispositivos y encriptación de datos, es un tema crucial en el ámbito de los circuitos integrados. De aquí nace la necesidad de desarrollar técnicas criptográficas eficientes que garanticen la seguridad y la autenticación de los equipos.

Tradicionalmente, popularizó el uso de dos técnicas de cifrado:

1. Random Number Generators (RNG).

Algoritmos aleatorios que evitan que los dispositivos sean atacados debido a su carácter impredecible. La creación de claves aleatorias suele obtenerse del muestreo de fenómenos físicos caóticos como por ejemplo el ruido térmico. [4]

2. Public Key Cryptography (PKC).

Técnica de cifrado que usa un algoritmo que empareja una clave pública y otra privada. El mensaje se encripta con la clave privada y se autentica con la clave pública.

No obstante, la inmensa mayoría de dispositivos de la actualidad establecen una serie de limitaciones de área, energía y recursos muy restrictivos. Esto hace que los algoritmos y métodos criptográficos clásicos supongan un coste prohibitivo.

Como método alternativo surgió la idea de usar como técnica de cifrado lo que se conoce como funciones físicas inclonables (PUFs). Se define como PUF a la firma digital que genera cada dispositivo debido a las variaciones de carácter aleatorio e impredecible devenidas de los procesos de fabricación de semiconductores. Esta firma digital e inherente para cada equipo singular se obtiene a través de la medición de parámetros físicos tales como corrientes, capacidades o retrasos temporales en las señales.

Muchas configuraciones de PUF propuestas en la literatura utilizan módulos “ad hoc” para su implementación. En este trabajo, se explora la posibilidad de utilizar componentes ya presentes en el dispositivo a identificar.

1.1. Objetivos.

El objetivo principal es estudiar si es viable explotar las variaciones intrínsecas de los sensores (integrados en una FPGA) las cuales se deben a las imperfecciones durante los procesos de fabricación de los semiconductores. Asimismo, se controlará parcialmente mediante el uso de Ring Oscillators (también sujetos a las variaciones de fabricación) las condiciones del entorno de operación.

Con esto se quiere conseguir un sistema de autenticación más robusto en el que, a diferencia de los PUF convencionales, entran en juego dos fuentes diferentes de información para la identificación de dispositivos:

1. Los sensores on-chip de la FPGA que miden parámetros físicos tales como temperatura o tensión.
2. Los Ring Oscillators que aunque son parte de las condiciones de entorno, podemos controlarlos como si fuesen entradas al sistema.

Para conseguir todo esto, se han establecido los siguientes objetivos parciales:

- Controlar los sensores on-chip de la FPGA.

Esto conlleva familiarizarse con el XADC que es una herramienta incorporada en las FPGA de Xilinx que permite sensar distintos parámetros físicos del dispositivo.

- Implementación de Ring Oscillators.

Creación de una XDC macro para la implementación y organización de las múltiples instanciaciones de ROs durante toda la FPGA.

- Adquisición y tratamiento de datos.

Controlar la transmisión de datos a través de una UART con comunicación vía puerto serie. Asimismo, adecuar los datos recibidos de manera que sean útiles para el posterior análisis de los mismos.

- Análisis de los resultados.

Estudio de la información obtenida del sistema durante distintos TEST de evaluación y conclusión sobre la calidad de la configuración de PUF desarrollada a lo largo del trabajo.

1.2. Organización del documento.

El presente trabajo se encuentra dividido en diferentes subsecciones, cada una de ellas abordando distintos aspectos y contribuyendo a diferentes fines.

- **Epígrafe 1: Introducción.**

Sirve como preliminar para familiarizar al lector sobre los conceptos básicos del tema tratado y los objetivos marcados.

- **Epígrafe 2: Estado del arte.**

Se explica en detalle el concepto de PUF, sus propiedades, sus aplicaciones y su marco regulador. Sitúa al lector dentro del ámbito en cuestión de una manera más profunda.

- **Epígrafe 3: Sistema propuesto.**

Se detalla todo la configuración y montaje del sistema. Asimismo se precisan los procedimientos de creación de una PUF. También se explica cómo se han utilizado cada una de las herramientas software con las que se han obtenido los resultados correspondientes.

- **Epígrafe 4: Resultados experimentales.**

Se muestra cómo se han tratado los datos obtenidos y cada uno de los experimentos realizados para dilucidar la posibilidad de usar el sistema propuesto como una PUF con la calidad suficiente.

- **Epígrafe 5: Conclusiones.**

Se comentan y analizan los resultados experimentales más a fondo y se da una valoración final sobre la implementación realizada. Además, se proponen unas líneas de investigación futuras para continuar en este campo de actuación.

- **Epígrafe 6: Planificación y presupuesto.**

Se ofrece información minuciosa sobre el plazo y coste invertidos en el presente trabajo.

2. Estado del arte.

Una función física inclonable (PUF: 'Physically Unclonable Function') se define como una huella individual y única de cada sistema físico. Atienden a las variaciones físicas devenidas de la fabricación de elementos semiconductores. De hecho, como su propio nombre indica, son inclonables. Esto se debe a las impredecibles e incontrolables características inherentes que se les otorga durante la fabricación.

Una función física inclonable (PUF) puede ser considerada como un circuito integrado (IC), capaz de producir diferentes salidas para el mismo conjunto de entradas cuando se implementan en dispositivos distintos. Se puede tomar ventaja de esta propiedad para generar una firma inherente para dispositivos hardware.

Los circuitos PUF se diseñan para explotar la variabilidad de las tecnologías CMOS, debido a las imperfecciones de sus procesos de manufactura, las cuales dan lugar a alteraciones intrínsecas y que no siguen ningún patrón en las propiedades físicas y eléctricas de los circuitos integrados, tales como la resistividad de los metales o, incluso, la longitud de canal efectivo de los transistores (CMOS).

El interés por las funciones físicas inclonables (PUF) reside en sus posibles aplicaciones en el ámbito de la seguridad y criptografía.

"Las funciones físicas inclonables son el equivalente a las transformaciones matemáticas que, bajo excitación externa, pueden generar respuestas irreversibles. Sobrepasando a sus homólogos matemáticos, su complejidad física inherente les otorga resiliencia a ser clonados o revertidos. Cuando estos rasgos se combinan con su invariancia en el tiempo y su operación determinística, la necesidad de registrar sus respuestas puede ser atenuada. Esta característica esencial, los convierte en componentes críticos para un amplio espectro de aplicaciones de autenticación criptográfica, donde el almacenamiento de información sensible es limitada." [5].

De hecho, las PUFs emplean la ejecución de procesos aleatorios que generan a su vez respuestas impredecibles. Y aunque son procesos irreversibles debido a su complejidad, si el mismo sistema físico fuese sometido a los mismos estímulos, sería generada la misma respuesta. De esta manera, se puede decir que la interacción sistema-objeto es puramente determinista.

Sin embargo, la idea de estudiar los rasgos físicos intrínsecos para identificar objetos o sistemas no es algo nuevo. Es más, el campo de la biometría tiene más de un siglo de antigüedad. Por otro lado, el uso de patrones aleatorios se desarrolló en el campo de la economía para el control de identificación de billetes falsos. De esta manera, adaptando esta idea al mundo de la electrónica, surgió el concepto de las conocidas como 'physical one-way functions' de la mano de Pappu Srinivasa Ravikanth. Con los años, la idea de funciones inclonables se fue extendiendo y sus propuestas y escenarios denotaron cierta tendencia a construcciones de mayor integración. El interés por los PUF ha aumentado sustancialmente, convirtiéndolo en un tema candente dentro del mundo de la seguridad hardware [6].

Las respuestas de una PUF suelen envolver medidas físicas tales como temperatura o tensión. De esta manera, se puede extraer que pueden aparecen efectos secundarios de carácter ambiental-físico que pueden alterar el resultado. Por tanto, el hecho de que se aplique la misma función al mismo objeto no garantiza que se vaya a obtener el mismo resultado. La realidad es que depende del ambiente que rodee al objeto durante el proceso.

Para recapitular, una función física inclonable da como resultado para cada entrada (input) una salida (output). Dicha salida no influye únicamente del estímulo al que se ve sometido el sistema, sino que además las condiciones del entorno también afectan a la reproducibilidad y estabilidad de la respuesta. Esto último dificulta su uso directo para el cifrado de claves criptográficas. Sin embargo, para reconocer que la salida obtenida corresponde al dispositivo en cuestión, es suficiente con una alta similitud entre los resultados.

Las funciones físicas inclonables (PUF) son consideradas un método de bajo coste para la autenticación de dispositivos, generar clave criptográficas y almacenarlas así como la generación de números aleatorios.

Con la explosión de los dispositivos IoT ("Internet of Things"), los PUF ofrecen numerosas oportunidades como técnica de seguridad Hardware.

IoT consiste en la interconexión de numerosos dispositivos con recursos limitados como redes de sensores y actuadores, los cuales van vinculados a Internet. La interconexión de estos dispositivos brinda la capacidad de recopilar una inmensa cantidad de datos para procesar y analizar. De hecho, una más que significativa parte de esos datos responden a información privada la cual no ha de ser interceptada o falsificada de ninguna manera. Es decir, la seguridad en los dispositivos IoT es primordial para el ulterior desarrollo de la tecnología. Y es aquí donde entran en juego las funciones físicas inclonables. [7]

Las técnicas de seguridad IoT han de superar las restricciones de área y energía que estos dispositivos suelen tener. Las soluciones Hardware basadas en PUF que se proponen son:

- Autenticación: para establecer confianza con un dispositivo IoT, éste ha de tener verificada su identidad. Este método se basa en almacenar en una base de datos segura la respuesta del dispositivo ante estímulos aleatorios. Y posteriormente, durante su verificación se comprueba que esa respuesta única e inherente del elemento corresponde con la registrada.

- Encriptación/Desencriptación: los algoritmos de cifrado suelen utilizarse para lograr garantizar privacidad y confidencialidad. Una aplicación de los PUF muy atractiva para este ámbito es la generación de claves de cifrado que evitan la necesidad de almacenar claves en el chip. Esto hace que las redes IoT sean menos susceptibles a ser asaltados por ataques de canal lateral.

En definitiva, la seguridad de estos dispositivos es de vital importancia para su implantación y utilización. No obstante, la inmensa mayoría de equipos IoT son ligeros y de bajo consumo, lo cual hace que los algoritmos clásicos de cifrado sean prohibitivamente costosos. Por tanto, las PUF son candidatos ideales para soluciones de seguridad asequibles para redes IoT debido a su relativamente sencilla estructura.

Dando soporte para solucionar los problemas de fiabilidad y seguridad de las que sufren las tecnologías PUF, hay mucho trabajo por hacer en este campo de investigación considerando las rigurosas restricciones de energía y recursos de los equipos IoT.

2.1. Conceptos básicos de PUF:

En este apartado se procederá a explicar ciertos conceptos básicos sobre funciones físicas inclonables (PUF). La familiarización con estas ideas clave será útil para comprender posteriormente sus propiedades y aplicaciones.

Estímulos y Respuestas.

Dada la condición de función de las PUF, estas ofrecen una respuesta ("Response") cuantificable al ser sometidos a un estímulo ("Challenge").

Es importante hacer hincapié que en la mayoría de casos, una PUF no es verdaderamente una función desde el punto de vista matemático, dado que un estímulo puede generar varias respuestas posibles. Por tanto, lo más apropiado es considerar una PUF como un proceso el cual actúa sobre un sistema físico particular.

Es típico llamar al estímulo y su respuesta correspondiente como Par Estímulo-Respuesta (CRP). Además, dentro de un escenario de actuación común, una PUF se usa en dos distintas fases:

1. Registro ("Enrollment"): se recopila una cantidad de CRP para una configuración de PUF particular y se almacenan en una base de datos CRP.

2. Confirmación ("Verification"): se somete al sistema a un estímulo previamente registrado en la base de datos. La respuesta generada se compara con la de la base de datos a modo de verificación.

Hamming Distance

La aplicación más común de las PUFs es la identificación de dispositivos. Para dicho fin, existen dos tipos de medidas:

1. Inter-Distancia: la que existe entre las respuestas de diferentes instanciaciones de PUF para el mismo estímulo.

2. Intra-Distancia: la que existe entre diferentes respuestas generadas por el mismo sistema para el mismo estímulo.

Cabe resaltar que para ambas medidas se aplica el mismo estímulo. Normalmente se usa como unidad de medida la distancia de Hamming. Ésta se expresa como una fracción entre la distancia de las "strings" a considerar, y la distancia de Hamming relativa.

La distancia relativa debería estar lo más cerca posible al 50% para poder determinar la mayor diferenciación posible. [6]

Condiciones de entorno de operación.

Partiendo de la base de que para la obtención una respuesta de una PUF se necesita medir un parámetro físico, hay que tener en cuenta efectos colaterales de segundo orden provenientes del entorno de operación que podrían interferir en la generación de una firma digital.

Tal y como se señaló en el apartado anterior "Hamming Distance", en un PUF un mismo estímulo no tiene necesariamente que desembocar en la misma respuesta dada la intra-distancia entre respuestas. La explicación a este efecto es el ruido aleatorio y la incertidumbre en la medida que, de forma inevitable, conlleva a perturbaciones en la medida.

Por otra parte, existen factores ambientales que tienen un efecto sistemático en la medida de la respuesta, como por ejemplo tensión de alimentación o temperatura.

Algunos de los impactos de estos efectos del entorno dependen de ciertos de detalles de implementación de PUFs. Por tanto, se han desarrollado estrategias de implementación las cuales reducen la dependencia de las condiciones de operación sobre la PUF.

Finalmente, existe una técnica llamada compensación la cual se puede aplicar si los efectos afectan al dispositivo de manera pseudo-lineal. Lo que se hace es considerar la relación entre dos medidas simultáneas para obtener un resultado mucho más robusto.

2.2. Propiedades de las PUF:

2.2.1. Constructibilidad.

Define la facilidad de construcción y de instanciación de una configuración PUF. Dado que las PUFs son objetos físicos, su requisito de constructibilidad es que sea posible crearla bajo las leyes de la Física. Desde un punto de vista más práctico, la "facilidad" de construcción se basa en el coste de instanciar una clase particular de PUF.

Resulta interesante destacar que la inviabilidad de una PUF está directamente relacionada con la cantidad de restricciones de su comportamiento CRP. [6]

2.2.2. Evaluabilidad

Una PUF se define como evaluable si se puede construir y si para cualquier instanciación aleatoria de la misma y cualquier estímulo se obtiene una respuesta.

Dentro de un marco práctico, una evaluación sencilla es aquella realizable dentro de unas restricciones estrictas de tiempo, área, energía y coste para una determinada aplicación. Es decir, que una evaluación puede resultar "sencilla" para una aplicación, mientras que para otra puede ser inviable. [6]

2.2.3. Reproducibilidad.

Una PUF es reproducible si es evaluable y si su intra-distancia de Hamming es ínfima.

La reproducibilidad es una propiedad de naturaleza práctica que viene dada por su comportamiento CRP. Se basa en que una PUF es reproducible cuando al someter al sistema bajo los mismos estímulos, se obtienen unas respuestas suficientemente similares como para que se reconozca como el mismo dispositivo. [6]

2.2.4. Singularidad.

Una PUF es singular si es evaluable y si su inter-distancia de Hamming es elevada.

La singularidad representa la diferenciación apreciable entre las respuestas de distintos dispositivos sometidos a los mismos estímulos bajo las mismas condiciones de entorno. [6]

2.2.5. Autenticación.

Una PUF es identificable si es reproducible y única.

El hecho de que una PUF sea identificable es el resultado de haber generado una firma digital inherente a cada dispositivo que lo diferencia del resto. Bajo unas condiciones y estímulos reproducibles, se ha de conseguir una respuesta altamente parecida para el mismo dispositivo. [6]

2.2.6. Inclonable.

Una PUF es inclonable si es evaluable y si es difícil de influir en su proceso de creación.

La dificultad de influir en su proceso de isntanciación refleja la imposibilidad física y técnica de generar dos instanciaciones iguales. Combinando los conceptos de constructibilidad e inclonabilidad se puede resumir que es fácil crear un sistema físico aleatorio, pero es extremadamente complicado crear uno en concreto. Esto se debe a las variaciones aleatorias e impredecibles de fabricación. [6]

2.2.7. Impredecible.

Una PUF es impredecible si es evaluable y es difícil pronosticar su próximo resultado.

Esta propiedad es bastante similar a la de inclonabilidad, salvo que aquí no se reivindica la variación entre distintos dispositivos, sino la variación aleatoria para el mismo circuito.

En el caso ideal, se puede demostrar que las respuestas ante los estímulos son completamente independientes. Esto significa que no se pueden predecir los resultados a través de ningún algoritmo. [6]

2.3. Métricas de PUF:

Existen diversos criterios que dan lugar a la clasificación de las PUFs acorde a su calidad. Algunos ejemplos de estas directivas de calidad son:

2.3.1. Singularidad (“Uniqueness”)

Mide la capacidad de un dispositivo para generar una firma de identificación única. A modo de estimar este valor, se usa lo que se conoce como distancia de inter-Hamming entre un conjunto de respuestas obtenidas por la misma arquitectura de PUF en diferentes equipos. [7]

$$(Uniqueness) = \frac{1}{\binom{l}{2}} \sum_{i=1}^{l-1} \sum_{j=i+1}^{l} HD(R_i, R_j) * 100\%$$

Donde tenemos que R_i y R_j son las diferentes respuestas generadas por los dispositivos ‘i’ y ‘j’ para la misma configuración de PUF. En el caso ideal, la singularidad alcanzaría el valor de 50% ,que significa que cada dispositivo genera una respuesta única.

2.3.2. Fiabilidad (“Reliability”)

Mide la capacidad de una función física inclonable (PUF) de generar una respuesta consistente ‘R’ para un estímulo ‘C’, dadas las variaciones de entorno de operación tales como tensión de alimentación o temperatura. A modo de estimar este valor, se usa lo que se conoce como distancia de intra-Hamming. [7]

$$(Reliability) = 100 - \frac{1}{n \times (l-1)} \sum_{j=2}^{l} \sum_{k=1}^{n} HD(R_{i,k}, R_{j,k})$$

Donde tenemos que $R_{i,k}$ y $R_{j,k}$ son los ‘k’-bit de las respuestas ‘i’ y ‘j’ del mismo dispositivo para la misma configuración de PUF. Por otro lado, ‘n’ representa el número de bits de la respuesta y ‘l’ el número de dispositivos. En el caso ideal, la fiabilidad sería del 100%, que significaría que la respuesta del PUF se mantiene independientemente del ruido y la aleatoriedad de las variaciones de las condiciones de entorno de operación.

2.4. Aplicaciones de PUF:

2.4.1. Autenticación e de dispositivos.

Como ya se ha mencionado anteriormente en este trabajo, la firma digital que genera una función física inclonable sobre un dispositivo es inherente al mismo e imposible de replicar.

Actualmente, las PUF se consideran un método de autenticación para circuitos integrados que pueden suplantar a las costosas técnicas de cifrado. El uso de funciones físicas inclonables para autenticar dispositivos es muy útil en entornos con recursos limitados en términos de área y consumo como por ejemplo FPGA, esto es, casos en los que los métodos criptográficos convencionales resultarían en un coste muy elevado.

Por tanto, una vez almacenada la respuesta de un dispositivo ante una serie de estímulos, si se quiere verificar la identidad del mismo solo habrá que someterlo a los mismos estímulos ("Challenges") bajo las mismas condiciones y comprobar lo similar que resulta la nueva respuesta ("response") con respecto a la registrada. [8]

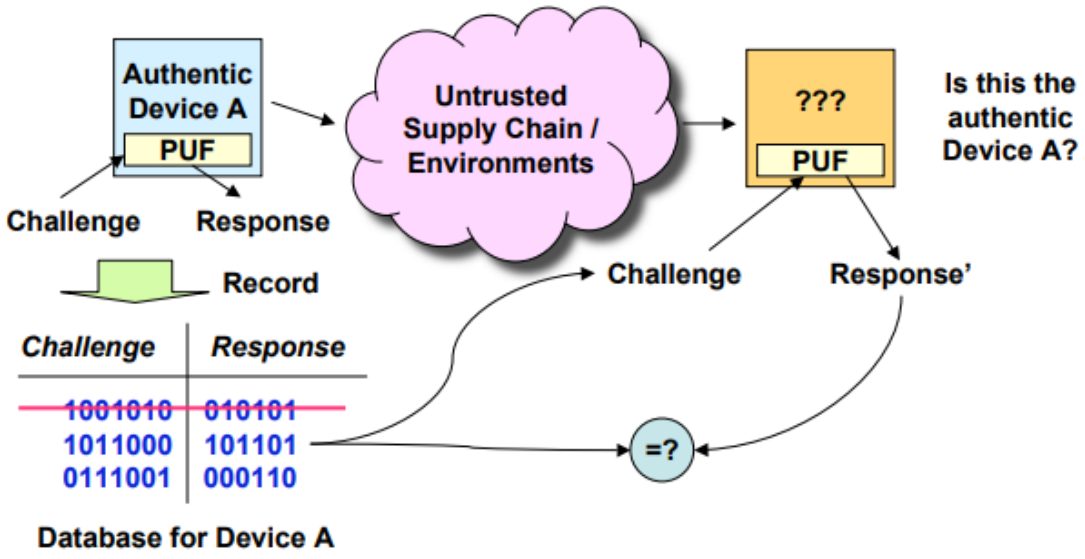


Figura 1: Autenticación de dispositivo mediante PUF [8]

2.4.2. Generación de claves de cifrado.

Para asegurar que sean impredecibles, las claves de cifrado deberían generarse por una fuente aleatoria. Desafortunadamente, las salidas ("outputs") de un PUF no son apropiados debido a que el ruido puede hacer que varíe ligeramente el resultado en cada evaluación, incluso utilizando el mismo circuito integrado y sometiéndolo a los mismos estímulos. Debido a esto, las salidas de las funciones físicas inclonables han de ser lo suficientemente estables para poder emplearlas para este objetivo.

Para conseguir claves de cifrado fiables a partir de generadores a los que le afecta el ruido como es el caso de las PUFs, se diseña lo que se conoce como "Fuzzy extractors". La función principal de un "fuzzy extractor" es convertir esas claves aptas para entornos en los que un mismo dispositivo puede generar valores similares pero no idénticos (Esto resulta muy útil en el campo de la biométrica).

Esta implementación usa la distancia de "Hamming" como su error umbral para generar con una alta probabilidad la misma clave. De esta manera, se pueden conseguir claves de cifrado lo suficientemente fiables. [9]

2.4.3. Generación de números aleatorios.

La necesidad de generar números aleatorios en procesos de cifrado es ubicua. Las funciones físicas inclonables son ampliamente usadas para este fin. Se usa el circuito PUF centrándose en estímulos meta-estables. La respuesta de una PUF es completamente impredecible, y además, varía en función de las condiciones de entorno a parte del estímulo en sí.

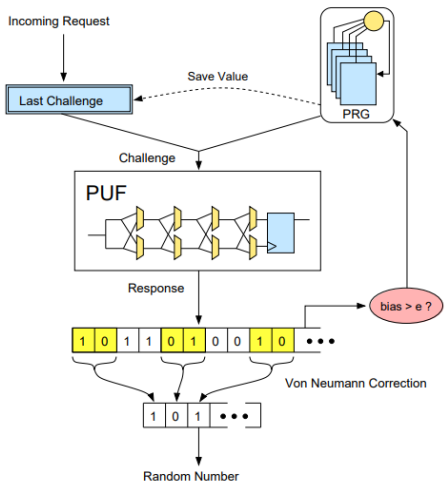


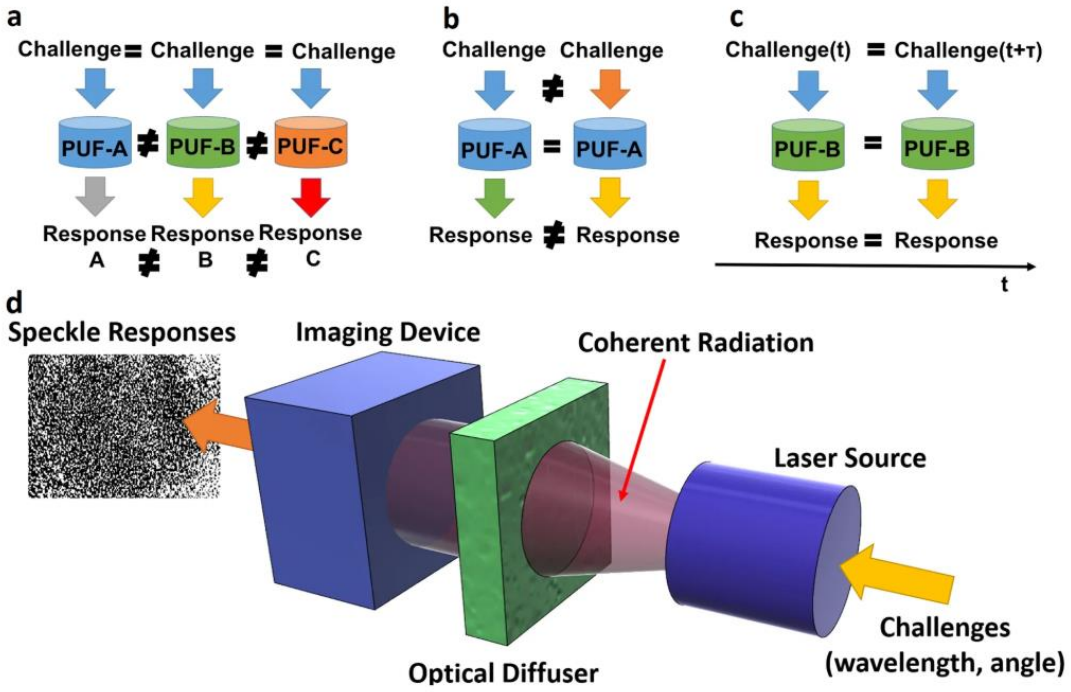
Figura 2: Generación de números aleatorios mediante PUF [7]

No obstante, los problemas de estabilidad de las PUF son un desafío a superar para este tipo de arquitecturas de generación de números aleatorios. [10]

2.5. Tipos de PUF:

2.5.1. Optical PUF:

Se incluyen dentro de este grupo todas aquellas funciones en las que la respuesta del sistema involucra un fenómeno óptico. Por lo general, suelen estar formados por un gran número de partículas de dispersión de luz inmersas en un medio óptico transparente. El resultado de atacar a dicho sistema con un rayo láser es aleatorio debido a la ubicación impredecible de las partículas.



Basic PUF properties: (**a**) unclonability output depends on the physical properties of the PUF, (**b**) unpredictability the output depends on the input, (**c**) time-invariant operation (robustness) (**d**) schematic of a typical optical PUF based on an optical diffuser.

Figura 3: Optical PUF [5]

2.5.2. Ring Oscillator PUF

Las PUFs basadas en “Ring Oscillators” explotan las variaciones en el retraso temporal de señales medibles debido a la variabilidad en los procesos de manufactura de equipos. De hecho, tratan la respuesta binaria de los componentes digitales como fuente de aleatoriedad que permita caracterizar la huella de dichos componentes. [6]

La salida se realimenta con su entrada, creando un bucle oscilante asíncrono llamado Ring Oscillator. De forma evidente, la frecuencia de oscilación viene determinada por el retraso temporal del bucle. Por tanto, medir la frecuencia de oscilación es equivalente a medir el retraso de la señal. Además, dicho valor de frecuencia es totalmente impredecible y aleatorio, debido a que depende de las variaciones en los procesos de fabricación de los semiconductores.

A continuación en la ‘Figura 4’, se podrá ver un diagrama de bloques que incluye el funcionamiento general de una función física inclonable basada en Ring Oscillator (ROPUF), identificando su par estímulo-respuesta (CRP):

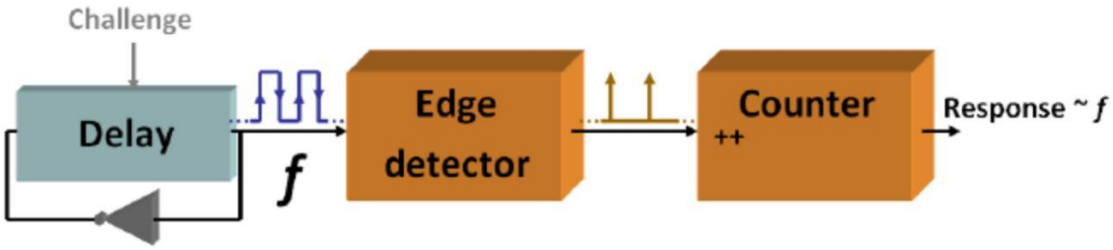


Figura 4: CRP de un ROPUF [6]

La obtención de la frecuencia de oscilación puede realizarse a través de componentes digitales como detectores de flanco o contadores.

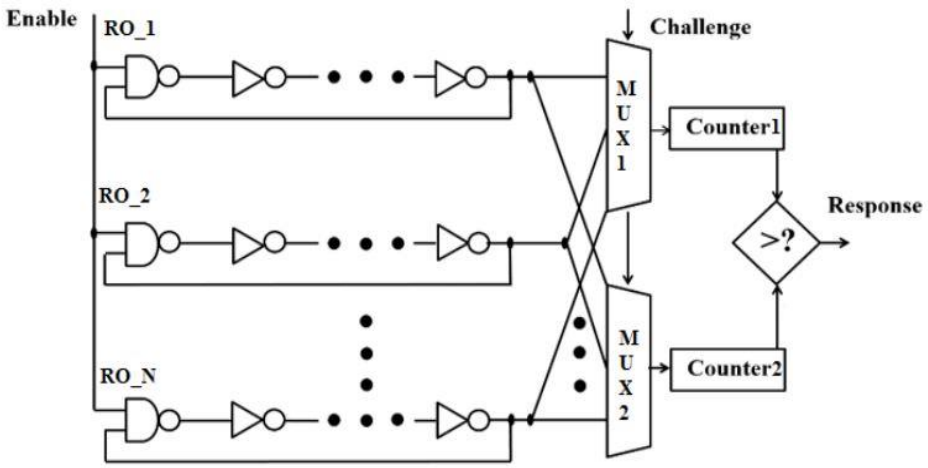


Figura 5: PUF genérico basado en Ring Oscillator [11]

2.5.3. Glitch-Based PUF:

Se basan en el uso de circuitos con lógica combinacional como 'glitches', es decir, como un error el cual no afecta negativamente al rendimiento del sistema.

Idealmente, los circuitos combinacionales no tienen estado interno, lo que significa que su respuesta en estado estacionario ('steady state output') viene determinada exclusivamente por sus señales de entrada. La incidencia del 'glitch' está influenciado por las diferencias de retraso temporal entre las señales de entrada y salida. Dado que esos retrasos son producidos por variaciones aleatorias, la aparición, la forma y el número de 'glitches' serán aleatorias para cada instanciación específica. Por tanto, la medida de ese comportamiento aleatorio puede ser utilizada como respuesta de una función PUF.

2.5.4. Arbiter PUF:

La idea de un arbiter PUF es establecer una "carrera digital" entre dos caminos en un chip y generar lo que se llama un circuito arbitrario que decide cuál de los dos caminos "gana la carrera". Si los caminos se han diseñado de forma simétrica (con el mismo retraso de señal programado), entonces el resultado es completamente impredecible.

Durante la producción del chip, las variaciones de fabricación tendrán un efecto físico que determinará el retraso de señal exacto para cada camino, generando un offset. Además, si ese offset es lo suficientemente pequeño, se comprometerán violaciones de tiempo de establecimiento (setup) y de retención (hold). Por lo tanto, la respuesta del sistema vendrá determinado por las variaciones aleatorias de fabricación. [6]

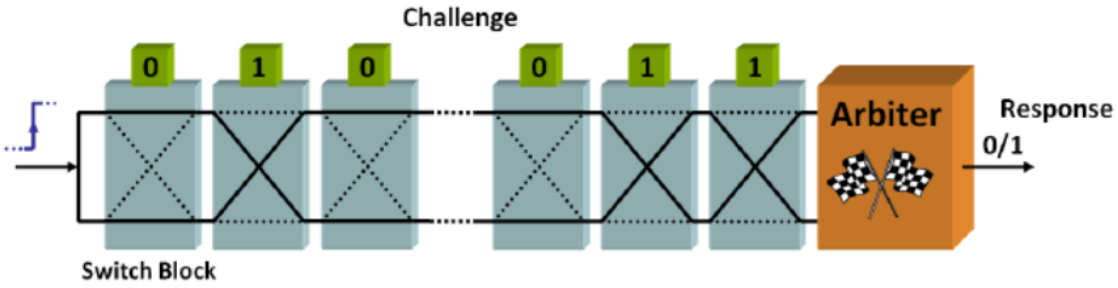


Figura 6: Operación básica de un Arbiter PUF. [6]

2.5.5. SRAM PUF:

El principio de operación que sigue una SRAM-PUF (Static Random-Access Memory Physically Unclonable Function) se basa en el uso del estado de aleatoriedad en el que entran algunas celdas de la SRAM justo después de entrar en funcionamiento.

Las celdas de SRAM tienden a posicionarse en un estado preferentemente (‘0’ o ‘1’) cada vez que ésta comienza a funcionar. Esto da como resultado variaciones aleatorias en las tensiones umbrales. De hecho, la arbitrariedad de la respuesta brinda un patrón único e imprevisible de ‘0’ y ‘1’. Este patrón se toma como la firma digital del dispositivo, la cual es inherente y por ende, diferente para cada chip. [12]

2.5.6. Sensor PUF.

Los sensores son componentes muy importantes en tecnologías como ‘Internet of Things’ (IoT) y que abarcan un gran abanico de aplicaciones.

Recopilar datos de forma segura usando los métodos de cifrado convencionales es una labor plagada de amenazas como la dificultad de privatizar la información y asegurar la confidencialidad de la misma. Asimismo, en aplicaciones de sensado se suele requerir que el sistema garantice cierto grado de veracidad y autenticidad de medidas.

La arquitectura general de un PUF basado en sensor trata de explotar la sensibilidad de la respuesta de un sensor. Esto permite que cualquier magnitud física se pueda convertir en un valor de tensión a través de un transductor on-chip. [13]

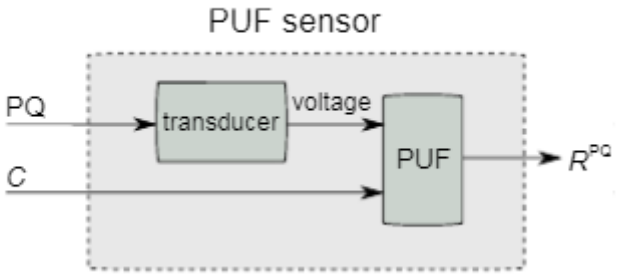


Figura 7: PUF sensor [13]

Las configuraciones de PUF convencionales cuentan con una entrada y una salida binaria. En cambio, los sensor-PUF cuentan con dos entradas, un estímulo binario y una magnitud física.

Un ejemplo trivial de sensor-PUF se encuentra en la dependencia que tienen los PUF convencionales con respecto a la temperatura. Es decir, la salida depende tanto del estímulo como de las condiciones de entorno (temperatura). [14]

2.6. Normativa y marco regulador.

Actualmente, el marco regulador en el que se engloba a las funciones físicas inclonables (PUF) se encuentra delimitado en el área de la ciberseguridad y a los estándar de autenticación de entidades.

- **ISO/IEC DIS 20897:**
  - **ISO/IEC DIS 20897-1:** Requisitos de seguridad para PUFs.
  - **ISO/IEC DIS 20897-2:** Pruebas y métodos de evaluación para PUFs.

- **ISO/IEC 9798:** Estándar para la autenticación de entidades que consta de:
  - **ISO/IEC 9798-1:** Especifica un modelo de autenticación y los requisitos y restricciones generales para los mecanismos de autenticación de entidades.
  - **ISO/IEC 9798-2:** Este documento especifica los mecanismos de autenticación de entidades usando algoritmos criptográficos de cifrado.
  - **ISO/IEC 9798-3:** En este documento se detallan un total de 10 mecanismos de autenticación de entidades que, basados en técnicas de encriptado asimétrico, resuelven una firma digital.
  - **ISO/IEC 9798-4:** Detalla aquellos mecanismos que usan funciones de verificación (‘check function’).
  - **ISO/IEC 9798-5:** Especifica aquellos mecanismos de autenticación que utilizan lo que se conoce como ‘zero-knowledge techniques’ las cuales suelen proporcionar autenticación unilateral.
  - **ISO/IEC 9798-6:** Referencia aquellos mecanismos basados en la transferencia de datos.

3. Sistema propuesto.

La configuración de PUF propuesto en este trabajo va más allá de cualquier cosa que se haya probado antes. Convencionalmente se habían propuesto 2 tipos de soluciones:

1. Sensar un parámetro físico y encriptarlo a través de un PUF.

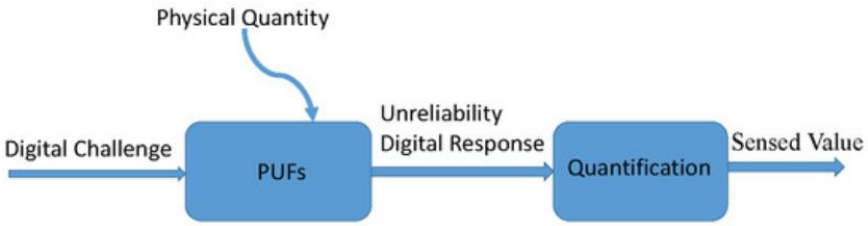


Figura 8: Diagrama de bloques de un PUF genérico para encriptar valores de sensado. [15]

2. Aplicar un Sensor-based PUF como el explicado en el apartado 2.6.5.

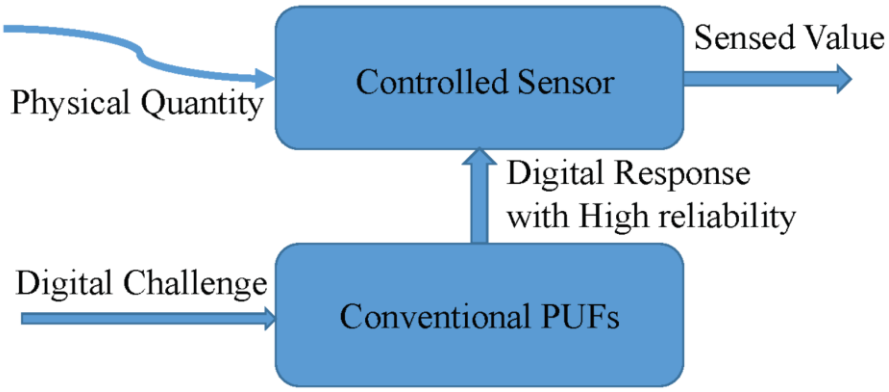


Figura 9: Diagrama de bloques de un Sensor PUF convencional. [15]

Sin embargo, en este caso se va a estudiar la posibilidad de explotar las variaciones intrínsecas de los sensores on-chip del XADC de distintas FPGA (dichas variaciones vienen dadas por los procesos de fabricación) y, al mismo tiempo, controlar parcialmente las condiciones de operación a través de la implementación de Ring Oscillator (los cuales están sujetos también a las variaciones de fabricación).

Por tanto, con la idea de esta propuesta se consigue aumentar significativamente la sensibilidad del PUF dado que las condiciones de entorno de operación le entran al sistema como un estímulo el cual puede ser controlado (Ring Oscillators).

De esta manera, el esquema del sistema propuesto como solución de PUF sería el siguiente:

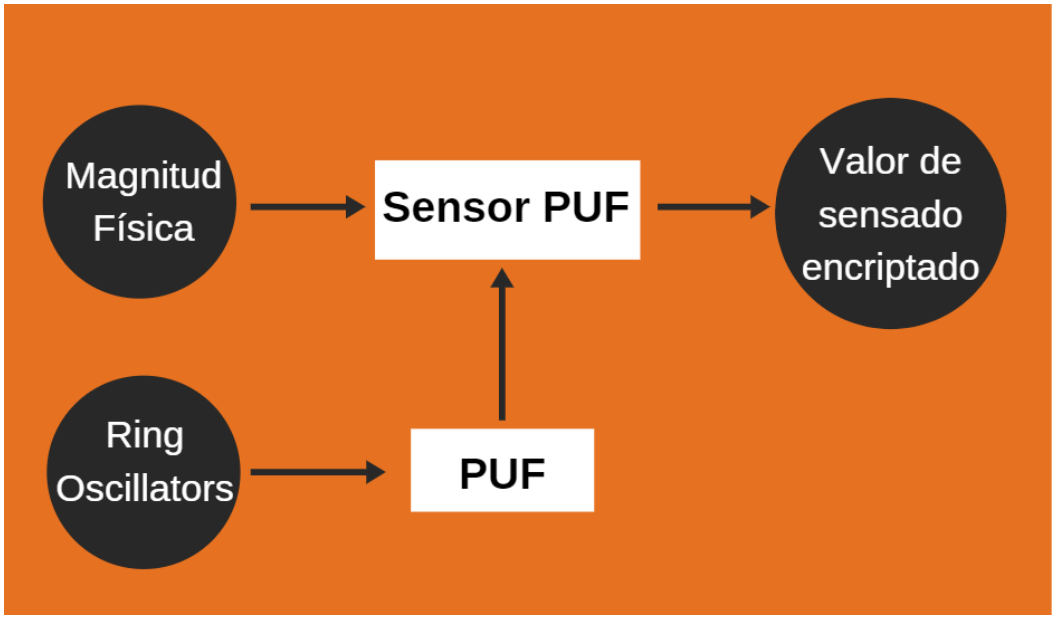


Figura 10: Diagrama de bloques de la configuración de PUF propuesta

A modo de ejemplo, se usará la temperatura como parámetro físico a medir. La configuración de un PUF cuenta con unas entradas conocidas como estímulo, una salida conocida como respuesta, y alrededor del sistema, existen unas condiciones de entorno las cuales influyen sobre la respuesta del sistema.

En este caso, la entrada del sistema sería la magnitud física (temperatura en este caso) como en cualquier PUF. No obstante, lo novedoso de esta propuesta es el control de los Ring Oscillators cuyo encendido y apagado provocan unas condiciones cambiantes en el entorno de operación, calentando y enfriando la FPGA. Esto es, añadimos una parte de las condiciones de entorno como estímulo (entrada) en sí mismo del sistema

SET-UP y configuración.

La idea de este PUF se basa en la autenticación de dispositivos. Para ello, serán usadas distintas FPGA las cuales lleven integrados sensores on-chip que nos permitan explotar las variaciones de fabricación a través de magnitudes físicas como la temperatura (T) y la tensión de core (VCCINT).

Por lo comentado anteriormente, se ha decidido optar por modelos de FPGA que cuentan con el módulo XADC, el cual será explicado más en detalle en el apartado 3.1.

Las distintas FPGA que se van a emplear durante los TEST de evaluación serán:

- Basys 3 Artix-7 FPGA Trainer Board (2 unidades).
  Part: *xc7a35tcpg236-1*.
- Arty Artix-7 FPGA Evaluation Kit (1 unidad).
  Part: *xc7a35tcsg324-1*.

De hecho, la idea inicial trataba de investigar si se podrían distinguir entre sí dos modelos iguales debido al principio de los PUF que dice que existen cambios a nivel de fabricación de los semiconductores. No obstante, se consideró interesante, además, estudiar la respuesta de otro dispositivo de la misma familia (Artix-7) pero distinto encapsulado.

Para la programación de las FPGA se ha usado el software de Xilinx Vivado 2018.3, siendo VHDL el lenguaje de programación utilizado. Para la implementación del sistema se han diseñado los siguientes módulos VHDL:

- XADC: este módulo sirve para acceder a los sensores del conversor analógico digital que las FPGA utilizadas llevan integrado. Posteriormente en el apartado 3.1 de este proyecto, se detallará la relevancia de este bloque. Además, ha sido considerado interesante adjuntar el código de programación de este bloque en el 'Anexo 3'.

- RAM: este bloque se encarga de registrar 15 medidas del XADC para cada caso. Este bloque de memoria refresca sus datos una vez está lleno y todos sus datos han sido comunicados vía puerto serie.

- UART: este bloque sirve para transmitir los datos del XADC a través de puerto serie. Se usa una velocidad de transmisión de 115200 baudios. Además, se ha considerado relevante incluir el código de este bloque en el 'Anexo 2'.

- HIGHLOW: este bloque es un paso intermedio entre el bloque ‘RAM’ y el bloque ‘UART’. La necesidad de incluirlo viene dada porque la UART transmite datos de 8 bits mientras que la RAM almacena palabras de 16 bits. Por tanto, a la hora de pasarle los datos a la UART hay que dividir la palabra en 2 partes. En total, para cada medida se envían 3 palabras distintas a la UART:

  I. zona de RO (1 dígito hexadecimal = 4 bits) & medida (15 downto 12).
  II. medida (11 downto 3)
  III. número de RO (2 dígitos hexadecimales = 8 bits)

  Cabe mencionar que la palabra que se lleva desde el módulo del XADC hasta el bloque de RAM es de 16 bits, y que en la guía de usuario del XADC especifica que los datos de los sensores son de 12 bits. De ahí, que se cojan de los 16 bits, los 12 más significativos (que son los que se corresponden con la medida).

- CONTROL: este bloque es el que controla el sincronismo y el funcionamiento de todos los demás bloques.

  I. Recibe del bloque XADC la confirmación de que se ha terminado la conversión y se puede registrar en el bloque RAM.
  II. Controla los tiempos en los que el bloque RAM lee la muestra del XADC y los que transmite la palabra al bloque HIGHLOW.
  III. Recibe desde el bloque UART cuándo está disponible para ser utilizada, y da la señal de habilitación para la transmisión de una nueva palabra.
  IV. Monitoriza la secuencia de encendido y apagado de “Ring Oscillators”.

- Ring Oscillator: este bloque es independiente de los demás. Recibe del bloque CONTROL la señal de habilitación del bucle combinacional. Es un bloque el cual su entrada depende directamente de su salida, y esta no sale a ningún otro punto del sistema. De hecho, el software lo toma como un error ya que se sale de lo convencional dentro de la programación VHDL. Posteriormente, en el apartado 3.2 se detallará más información sobre este bloque. Además, se ha considerado interesante incluir el código de este bloque en el ‘Anexo 1’.

A continuación, en la 'Figura 11', se puede observar el diagrama de bloques que representa todos los módulos VHDL empleados para la implementación del PUF. Sobre todo, es interesante darse cuenta de cómo el bloque de Ring Oscillators es el único que no cuenta con una señal de reloj (clk) que permita que sus procesos se realicen de forma síncrona (por tanto, es un bucle combinacional).

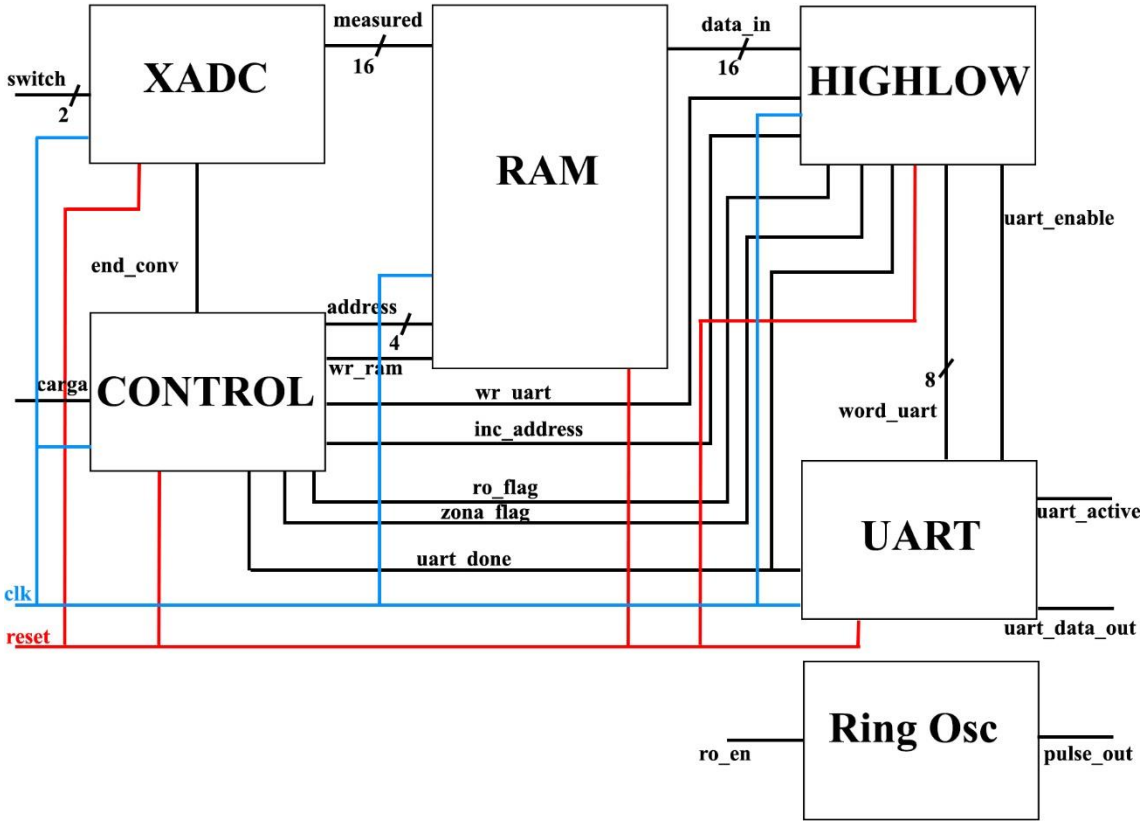


Figura 11: Diagrama de bloques. Programación VHDL.

3.1. Xilinx Analog to Digital Converter (XADC).

La decisión de utilizar esos modelos de FPGA vino condicionada con el hecho de que ambas tuvieran integrados un convertidor analógico-digital (XADC).

A través de la configuración de una IP personalizada desde el software Vivado de Xilinx, se ha decidido que este conversor trabaje leyendo muestras de forma continua (‘continuous conversion mode’) y se han deshabilitado todas las opciones de promediado y calibración para que las muestras representen fielmente lo que está ocurriendo en el dispositivo.

Este conversor es de 12 bits y da información sobre temperatura y tensión a través de sensores “on-chip” como por ejemplo:

- Temperatura del núcleo (T).
- Tensión del núcleo (Vccint).
- Tensión de alimentación auxiliar (Vccaux).
- Tensión de bloques de memoria RAM (Vccbram).

Los datos recibidos por estos sensores serán tratados de forma que se pueda saber si cada unidad tiene una “huella digital” independiente, si existe algún tipo de correlación entre ellos, y de más.

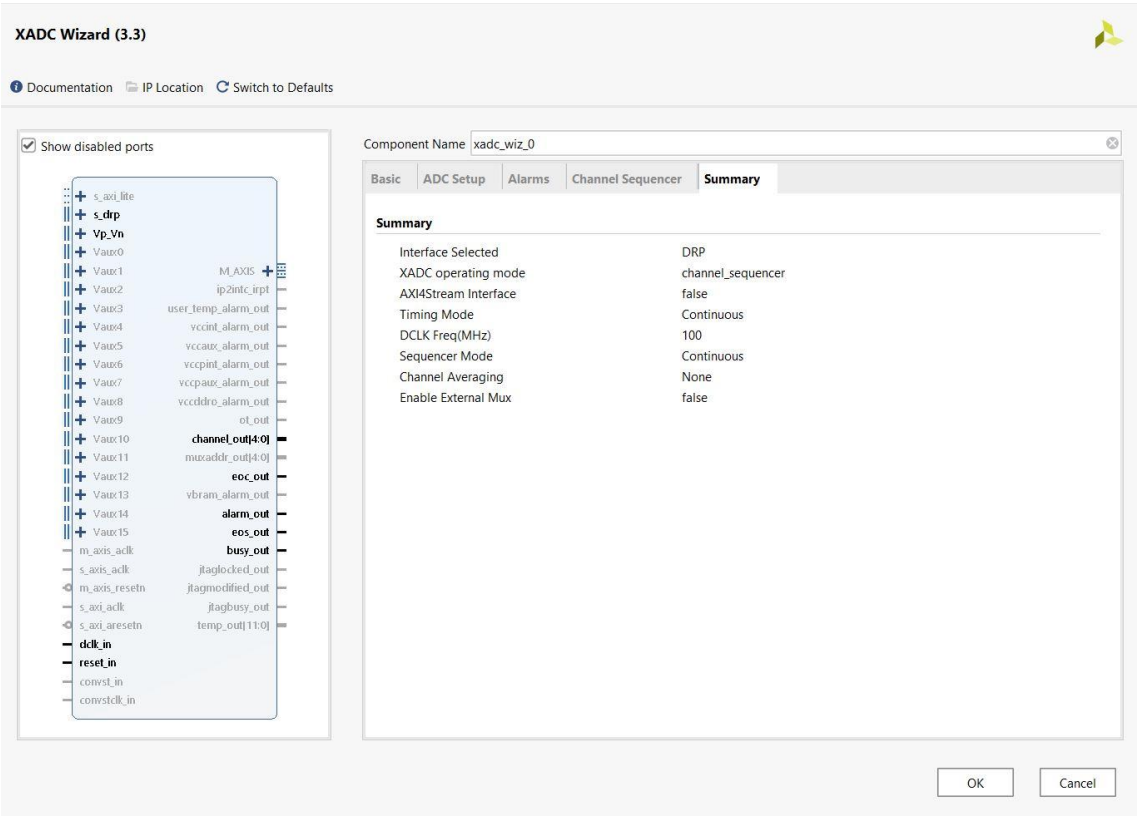


Figura 12: Configuración de la IP del XADC desde Vivado 2018.3

3.2. Ring Oscillators.

Un 'Ring Oscillator' se compone de un número impar de puertas lógicas NOT en cadena, retroalimentando la entrada con la salida. De esta manera, su salida oscila entre 2 valores de tensión. Además, en este caso, se ha añadido una señal habilitadora (enable) de forma que se pueda controlar el funcionamiento y el reposo de dichos ROs.

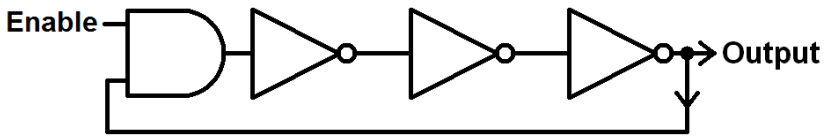


Figura 13: Circuito de Ring Oscillator

Tal y como se puede ver en la 'Figura 13', el circuito de un "Ring Oscillator" no deja de ser un bucle combinacional cuya entrada depende directamente de su salida. Y además, no cuenta con ninguna señal de reloj (CLK) que aporte sincronismo. Por tanto, cabe mencionar que la implementación de 'Ring Oscillators' escapa de la visión clásica de la programación VHDL.

En relación a lo comentado previamente, el propio software de Xilinx (Vivado) tomó la implementación de estos osciladores como error en primera instancia. De hecho, se ha tenido que recurrir al uso de restricciones ("constraints") para que se permitiera la implementación de los mismos. Asimismo, como se necesita incluir un comando para cada una de las instanciaciones de RO (en total son 2016 instanciaciones), se ha optado por programar un script en Matlab el cual escriba en un txt el siguiente comando, cambiando para cada instanciación su <hier> y su correspondiente <net>:

set_property ALLOW_COMBINATORIAL_LOOPS true [get_nets <myHier/myNet>];

Además, ha sido necesario el hecho de eliminar las optimizaciones de diseño predeterminadas por el software de Xilinx para que dejase intactos los bucles combinacionales durante el proceso de implementación.

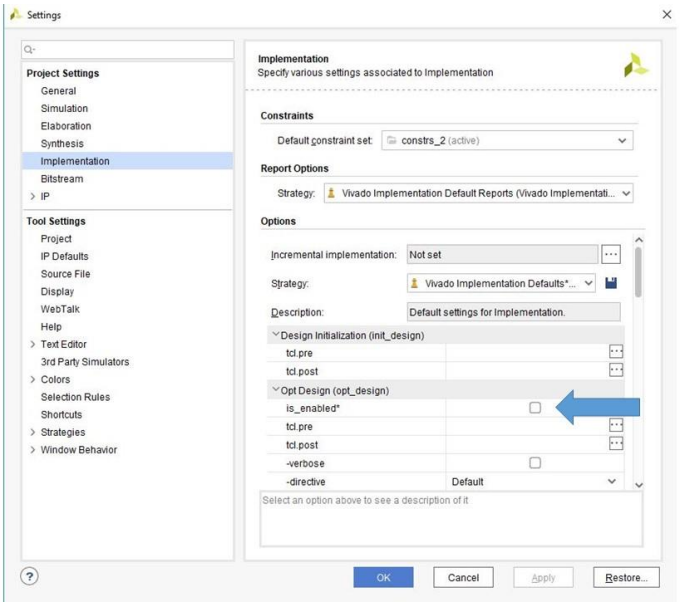


Figura 14: Configuración de optimizaciones de software Vivado 2018.3

Por otro lado, la herramienta de síntesis suele absorber señales durante sus procesos. La explicación es que cuando una señal pasa de un bloque lógico a otro, dicha herramienta le asigna un nombre (en la mayoría de casos aleatorio). De esta manera, no se podrá aplicar cualquier 'constraint' que se desee sobre la señal original. Asimismo, no se podrá encontrar dicha señal mediante ningún método de depuración.

Para lograr mantener dicha señal original, se debe usar el atributo 'KEEP' anteriormente mencionado. Como cualquier atributo en VHDL, primero debe ser definido y después se asocia con la señal que se desea mantener. Esto puede verse en el 'Anexo 1'.

3.3. Creación de una XDC macro e instanciaciones de los RO.

XDC macros definen la ubicación relativa de celdas usando coordenadas relativas. Además, las celdas correspondientes se asignan a la XDC macro bajo un patrón de ubicación relativo con respecto a la misma macro. Por tanto, la macro se puede colocar en cualquier parte del dispositivo si se mantienen las ubicaciones relativas especificadas entre las celdas.

La diferencia fundamental con respecto a los RPMs ('Relatively Placed Macros') es que estos usan atributos ('HDL attributes') y los XDC macros usan restricciones ('XDC constraints'). Por otra parte, un RPM es creado durante el proceso de síntesis, mientras que los XDC macros se crean durante la implementación. [16]

De hecho, esto último es una razón por la que en este proyecto se ha elegido usar XDC macros. La necesidad de manejar una macro viene dada por la instanciación de 2016 Ring Oscillators en la FPGA. Cada uno de los ROs cuenta con varias 'leaf cells' y 'nets' a la hora de su implementación. Por tanto, es conveniente desarrollar un método a través del cual manejar todas las instanciaciones del mismo módulo de manera eficaz y eficiente.

Para conseguir que los distintos LUTs de los diferentes ROs tuviesen el mismo nombre, y que de esta manera se pueda automatizar la exportación de las correspondientes 'constraints', hizo falta implementar los ROs como módulos fuera de contexto de la síntesis ("out-of-context for synthesis").

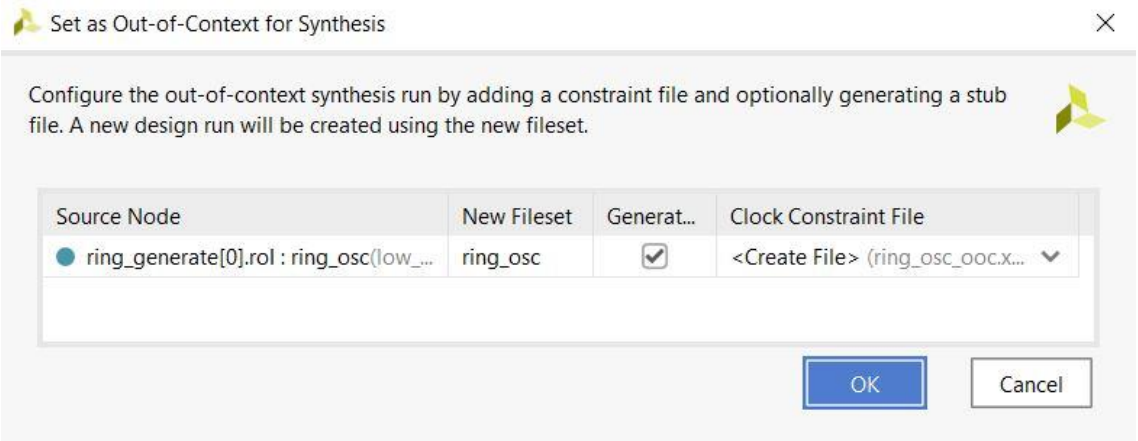


Figura 15: Implementación de ROs como módulos fuera de contexto.

Una vez realizada la implementación de los módulos VHDL, es hora de crear la XDC macro a través de los siguientes comandos utilizando la consola TCL: [17]

- *create_macro <name_macro>*

Este comando sirve para crear la XDC macro.

- *update_macro <name_macro> { <parent>/<leaf_cell > <Relative_position>}*

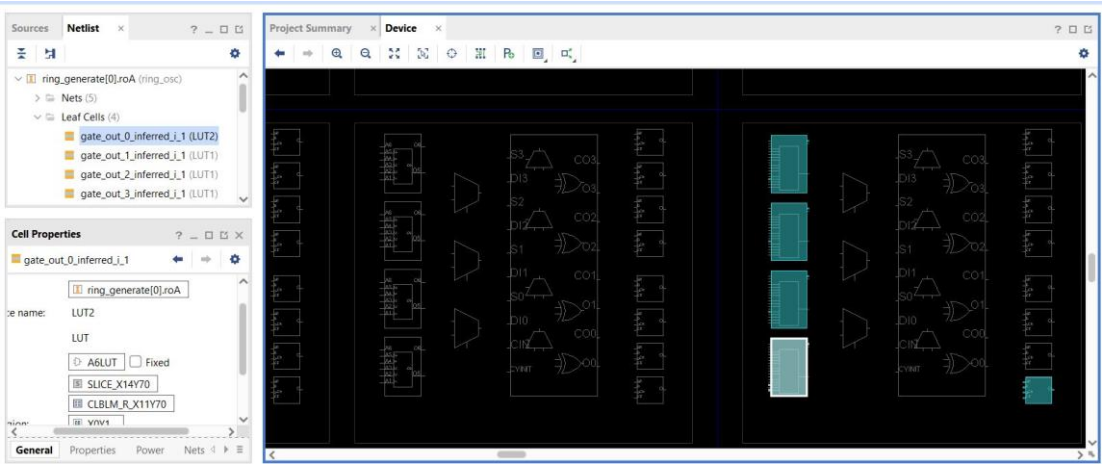


Figura 16: Leaf cells de un Ring Oscillator

En este caso, y como se puede ver en la 'Figura 16', cada RO tiene 4 leaf cells por tanto el comando quedaría de la siguiente manera:

update_macro macro { ring_generate[0].roA /gate_out_0_inferred_i_1 X14Y70
ring_generate[0].roA /gate_out_1_inferred_i_1 X14Y70
ring_generate[0].roA /gate_out_2_inferred_i_1 X14Y70
ring_generate[0].roA /gate_out_3_inferred_i_1 X14Y70}

- *get_cells –of [get_macros <name_macro>]*

Este comando sirve para confirmar que todas y cada una de las celdas que corresponden a un mismo RO se han implementado en la XDC macro.

- *write_xdc –cell [get_cells <parent>] <name_constraints_macro>.xdc*

Se crea un fichero .xdc el cual guarda la información sobre la macro creada y las celdas relacionadas a la misma.

- *read_xdc –cell [get_cells {<parent_2> <parent_3>… <parent_n>}] <name_constraints_macro>.xdc.*

Se usa para aplicar las 'constraints' de la XDC macro a todas y cada una de las instanciaciones del módulo de Ring Oscillator.

Debido a la gran cantidad de instanciaciones de Ring Oscillator que se implementan, se ha optado por diseñar un Script de Matlab a través del cual se rellen un txt con los comandos necesarios para la instrucción previa de 'read_xdc'. La programación de dicho Script de Matlab se podrá encontrar en el 'Anexo 4'.

A partir de este momento, ya se ha creado la XDC macro. Sin embargo, tal y como se puede apreciar en la 'Figura 17', las celdas han sido ocupadas de la manera en que el software ha elegido.

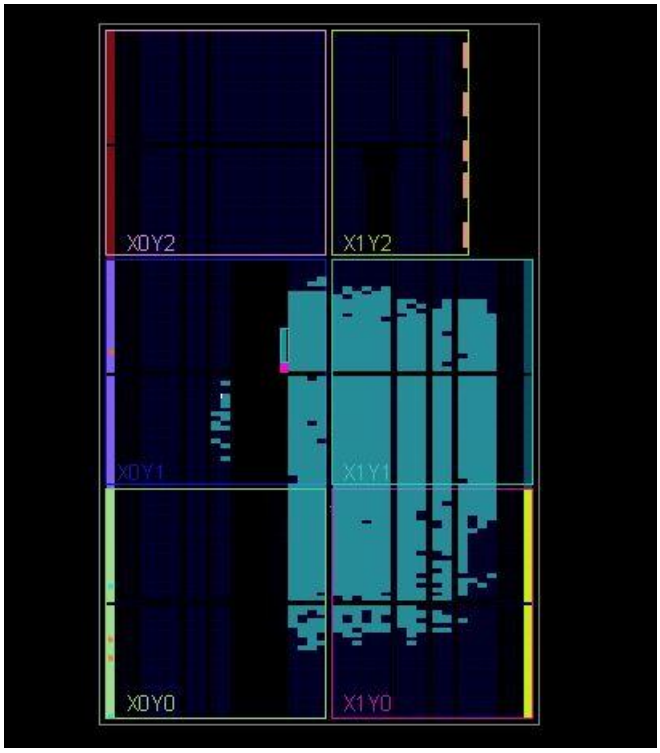


Figura 17: Arquitectura de la FPGA

Aunque la estructura interna de la FPGA no es simétrica, se ha diseñado la ubicación de los ROs de manera que se respete de alguna manera una simetría con respecto al centro de la FPGA. Para ello, se ha usado la siguiente 'constraint' para cada uno de los Ring Oscillator:

set_property LOC SLICE_<cell_location> [get_cells {<parent>/<leaf_cell>}]

Cabe destacar que esto funciona escribiendo cualquier 'leaf_cell' que contenga el RO dado que al haber creado una XDC macro, al mover una celda, se mueven todas. Además, al haber referido a los RO como módulos fuera de contexto durante la síntesis, las celdas de cada instanciación reciben el mismo nombre. Todo esto favorece la automatización de la exportación de restricciones ('constraints') y lo convierte en un método muy elegante dado que no hay que ir uno por uno modificando los comandos.

Después de la implementación, y con todas las ‘constraints’ necesarias, la estructura de la FPGA queda de la siguiente manera:

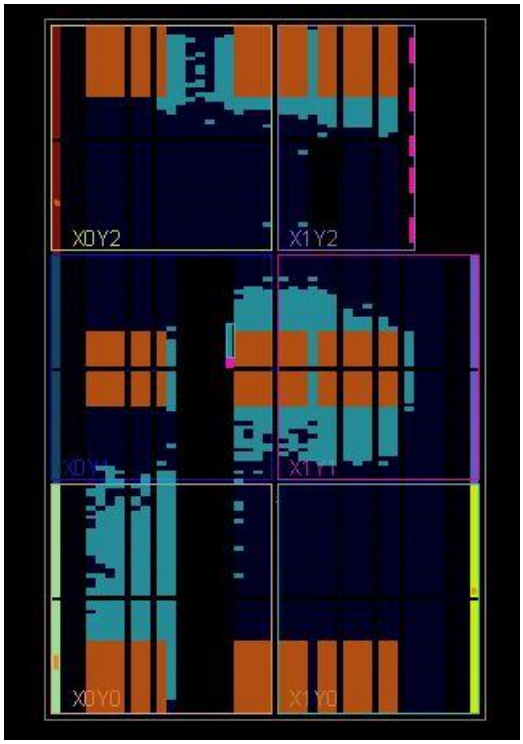


Figura 18: Distribución de las áreas de FPGA con Ring Oscillator.

Las celdas coloreadas de color naranja son aquellas en las que se sitúan los ROs. En total hay 9 zonas de 224 Ring Oscillators cada uno, lo que hace un total de 2016 ROs. Tal y como se ha comentado anteriormente, las dimensiones de la FPGA no son simétricas dado que la zona X1Y2 se ve acortada con respecto a X1Y1 y X1Y0.

Sin embargo, se ha dividido la estructura en un rectángulo de 150 celdas en el eje vertical (Y) y 58 celdas en el eje horizontal (X).

Finalmente, la secuencia del sistema ha sido configurada de tal modo que se van encendiendo de uno en uno todos los ROs de una zona, y cuando esta se completa, se apagan en orden de tal manera que el último encendido es el primero que se apaga. Finalmente, cuando se acaba con una zona se pasa a la siguiente, hasta que se finaliza con la novena zona (zona I) que es la que se sitúa en el centro de la FPGA.

3.4. Comunicación vía puerto serie.

Como se ha comentado anteriormente, hay 9 zonas, con 224 RO cada zona, 15 muestras por cada caso, y que se encienden y se apagan una vez cada uno, tenemos un total de:

$$\#muestras = 224 \cdot 9 \cdot 2 \cdot 15 = 60480\ muestras$$

Para cada una de esas 60840 muestras se escupen 6 dígitos hexadecimales a través de la UART. Estos 6 dígitos contienen la información de la medida del sensor, el RO que se ha encendido o apagado, y la zona a la que corresponde, de manera que, por ejemplo:

Dada la medida de temperatura 9BC para el RO número 123 de la zona B, tenemos que la codificación sería: "19BC7B".

Tabla 1: Ejemplo de codificación de medida en 6 dígitos.

1	9BC	7B
Zona B	Medida de 12 bits del sensor XADC	123 en Hexadecimal.

- Zonas: Hay 9 zonas A:I ordenadas alfabéticamente que corresponden a los números 0:8.
- Medida: Cada medida del sensor XADC cuenta con 12 bits.
- RO: Hay 224 Ring Oscillator 0:223 por cada una de las zonas ordenados en hexadecimal 0:DF.

Cada una de las muestras se transmiten por puerto serie a través de un cable USB y rellenan un archivo txt llamado "medida". Para este fin se utiliza la opción 'Capture' del software RealTerm tal y como se puede ver en la 'Figura 19'.

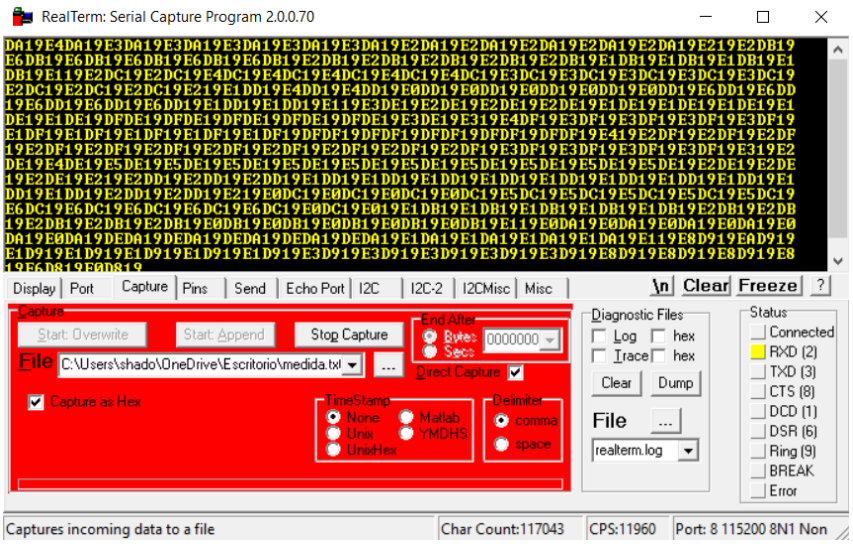


Figura 19: RealTerm. Comunicación vía puerto serie

3.5. Acondicionamiento de datos a través de código C++.

Como se puede sospechar por la 'Figura 19', un archivo txt en el cual todas y cada una de las medidas aparecen consecutivamente sin delimitadores, no es para intuitivo ni sencillo de estudiar.

Por tanto, a través del código C++ el cual puede encontrarse posteriormente en el 'Anexo 5' y 'Anexo 6', se procederá a tabular la información y a delimitarla por cada una de las distintas muestras que se toman. Para este fin se ha usado el software de programación DevC++.

De esta manera, se ha separado la información de manera que se obtengan los distintos archivos txt:

- 6digitos: Este archivo delimita cada medida con un salto de línea, manteniendo la zona, la medida y el RO de cada muestra consecutivos entre sí. La utilidad de este archivo es saber de una forma más intuitiva si la transmisión de datos se ha efectuado de manera no defectuosa y no se ha perdido ninguna muestra.

- Medida: Este archivo se queda únicamente con la medida de cada muestra delimitada con un salto de línea. A partir de este archivo se ha trabajado durante los Experimentos explicados durante el apartado 4. 'Resultados experimentales'.

- RO y Zona: Son dos archivos txt cuya estructura es la misma que la del anterior Medida, solo que estos guardan los bits que dan información sobre ROs y sobre las Zonas respectivamente. Al final estos txt no han aportado valor final y, por tanto, no aparecen ni se han utilizado durante el tratamiento de datos durante el apartado 4. 'Resultados experimentales'.

4. Resultados experimentales.

A partir del sistema presentado, se ha decidido tratar los datos obtenidos de acuerdo a 4 diferentes experimentos, a través de los cuales se pretende estudiar a fondo la respuesta de las FPGA ante distintos estímulos.

Para ello, se han tomado medidas de los sensores del XADC de cada FPGA durante 5 días. Durante cada uno de esos 5 días se ha ido cambiando el orden por el cual se adquirían los datos de cada FPGA. El efecto de esto cobra relevancia y se explicará durante el 'Experimento 2' el cuál será detallado posteriormente.

Tabla 2: Orden de disposición de las FPGA

Orden de disposición de las FPGA				
Día 1	**Día 2**	**Día 3**	**Día 4**	**Día 5**
Basys20	Basys28	Arty	Basys20	Basys28
Basys28	Basys20	Basys20	Arty	Arty
Arty	Arty	Basys28	Basys28	Basys20

Además, durante cada uno de esos 5 días se han adquirido datos 4 veces, cambiando a su vez el orden de adquisición de la información de cada sensor. El efecto de esta variación es útil durante el 'Experimento 3' el cual será detallado posteriormente.

Tabla 3: Orden de adquisición de datos

Orden de adquisición de datos			
TEST 1	**TEST 2**	**TEST 3**	**TEST 4**
Temperatura	Vccbram	Vccaux	Vccint
Vccint	Temperatura	Vccbram	Vccaux
Vccaux	Vccint	Temperatura	Vccbram
Vccbram	Vccaux	Vccint	Temperatura

Antes de comenzar a explicar detalladamente cada experimento y sus resultados, es conveniente recordar la disposición de los Ring Oscillator sobre la FPGA.

Se han generado 9 zonas designadas alfabéticamente {A, B, C, D, E, F, G, H, I} con un total de 224 ROs en cada una de ellas (formando un total de 2016 instanciaciones del módulo de Ring Oscillator). Tal y como se puede ver a continuación:

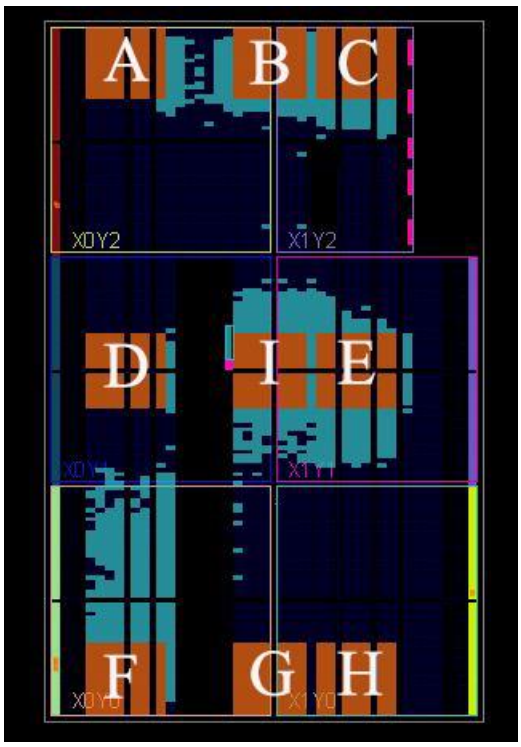


Figura 20: Zonas de ROs en la FPGA

Por otra parte, durante la adquisición de datos, la secuencia del sistema ha sido configurada de tal modo que se van encendiendo de uno en uno todos los ROs de una zona, y cuando esta se completa, se apagan en orden de tal manera que el último encendido es el primero que se apaga. Finalmente, cuando se acaba con una zona se pasa a la siguiente, hasta que se finaliza con la novena zona (zona I) que es la que se sitúa en el centro de la FPGA.

A parte de toda la información que se podrá extraer de cada uno de los siguientes experimentos, también se podrá saber si la FPGA es más sensible a la acción de los Ring Oscillator en determinadas zonas de la misma. De hecho, en primera instancia se cree que las zonas centrales tendrán un mayor impacto en los sensores del XADC, y las más alejadas tendrán un menor impacto. Sin embargo, todo depende de la posición en la que se encuentren dichos sensores dentro del chip.

Finalmente, es relevante mencionar que los datos de medida obtenidos hasta ahora están codificados en sistema hexadecimal. Para poder estudiar los resultados con valores reales de temperatura y tensión hay que convertirlos a través las fórmulas que se pueden encontrar en la Guía de Usuario del XADC. [18]

- Conversión de Temperatura:

$$Temp\ (°C) = \frac{ADC\ Code \times 503.975}{4096} - 273.15$$

- Conversión de Tensión:

$$Voltage = \frac{ADC\ Code}{4096} \times 3V$$

Donde tenemos que el ADC Code es la medida de 12 bits la cual ha quedado registrada para cada muestra en el archivo 'medida.txt' para cada TEST de evaluación.

4.1. Experimento 1: Comparativo entre las distintas FPGA.

En este experimento se hace una comparación entre las distintas FPGA para cada uno de los 4 parámetros a estudiar, a saber: Temperatura, Vccint, Vccaux y Vccbram.

Para ello, se mostrará una gráfica en la que se enfrentarán en el eje vertical el valor de los sensores del XADC, y en el eje vertical cada uno de los casos de encendido y apagado de los ROs. Teniendo en cuenta que hay 9 zonas, con 224 RO cada zona, 15 muestras por cada caso, y que se encienden y se apagan una vez cada uno, tenemos un total de:

$$\#muestras = 224 \cdot 9 \cdot 2 \cdot 15 = 60480\ muestras$$

No obstante, para que haya un único valor del eje Y para cada valor del eje X, se promediarán las 15 muestras que hay por cada caso, haciendo la media entre cada una de ellas.

$$\#casos = 224 \cdot 9 \cdot 2 = 4032\ casos$$

Para la realización de este experimento se ha diseñado un script de Matlab el cual puede encontrarse en el 'Anexo 7'.

- Temperatura:

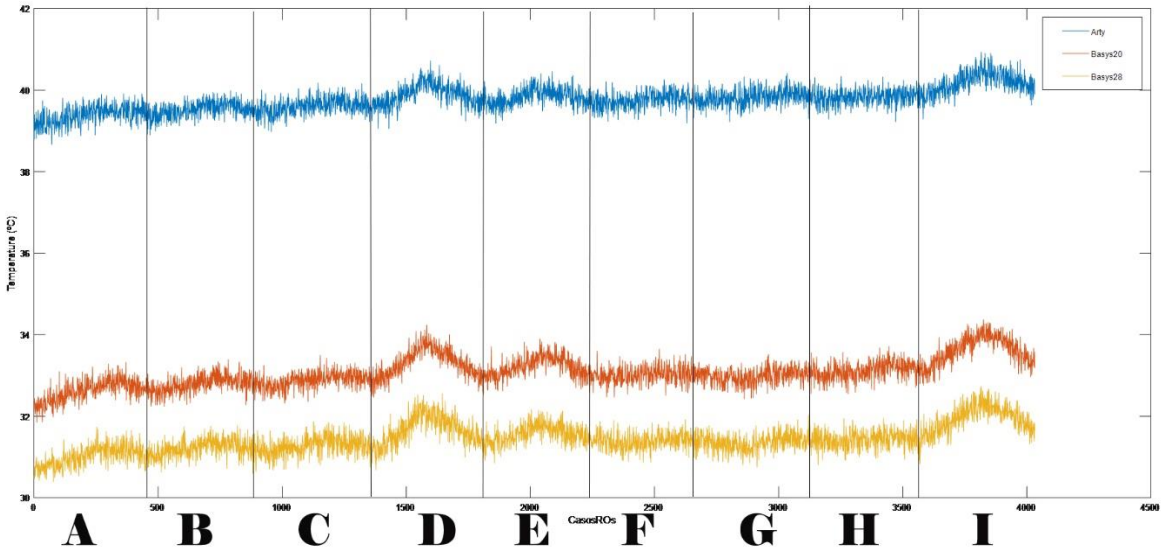


Figura 21: Gráfica de Temperatura del Experimento 1

- Vccint:

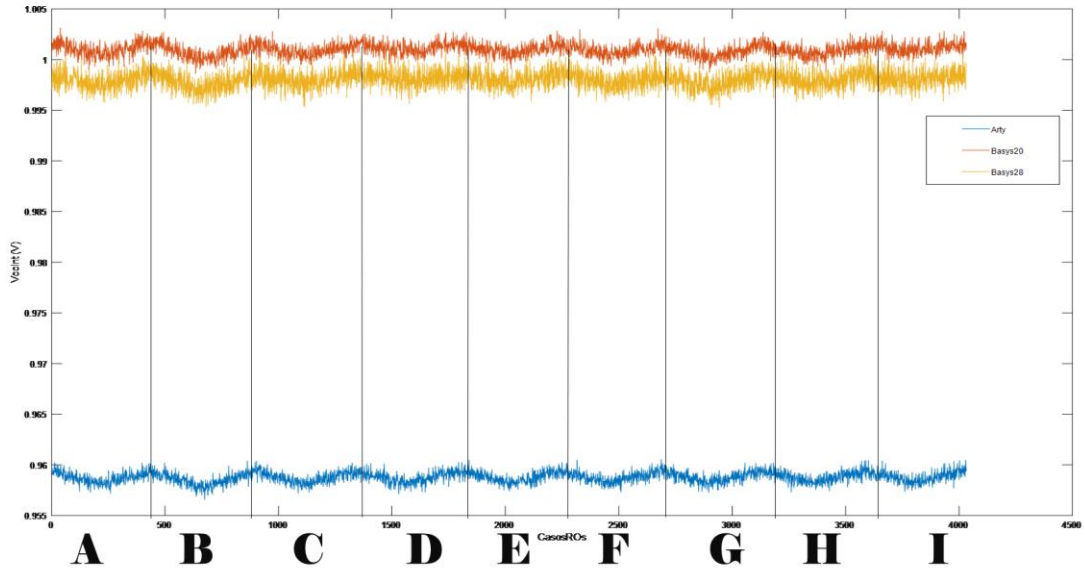


Figura 22: Gráfica de Vccint del Experimento 1

- Vccaux:

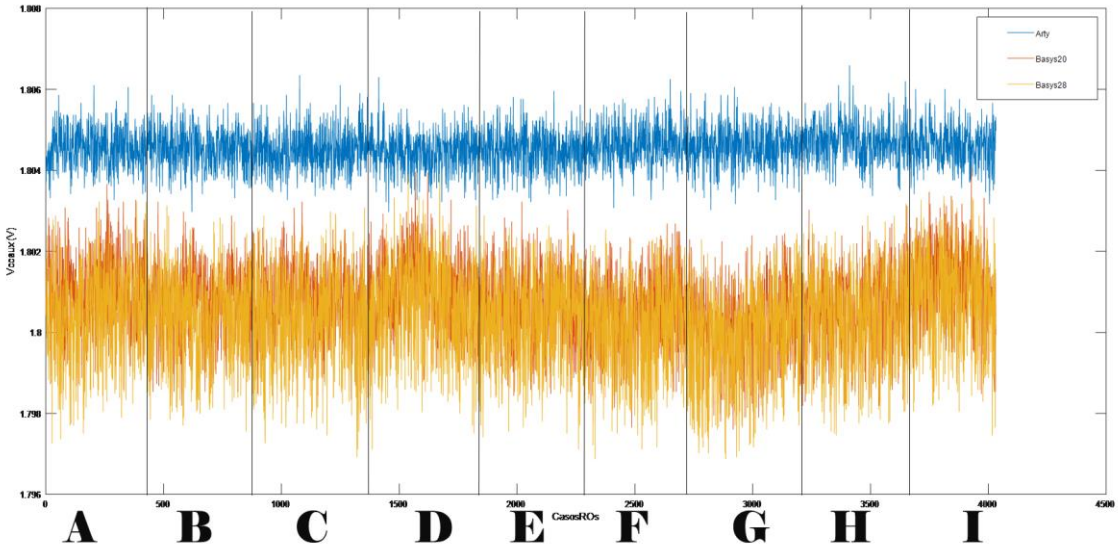


Figura 23: Gráfica de Vccaux del Experimento 1

- Vccbram:

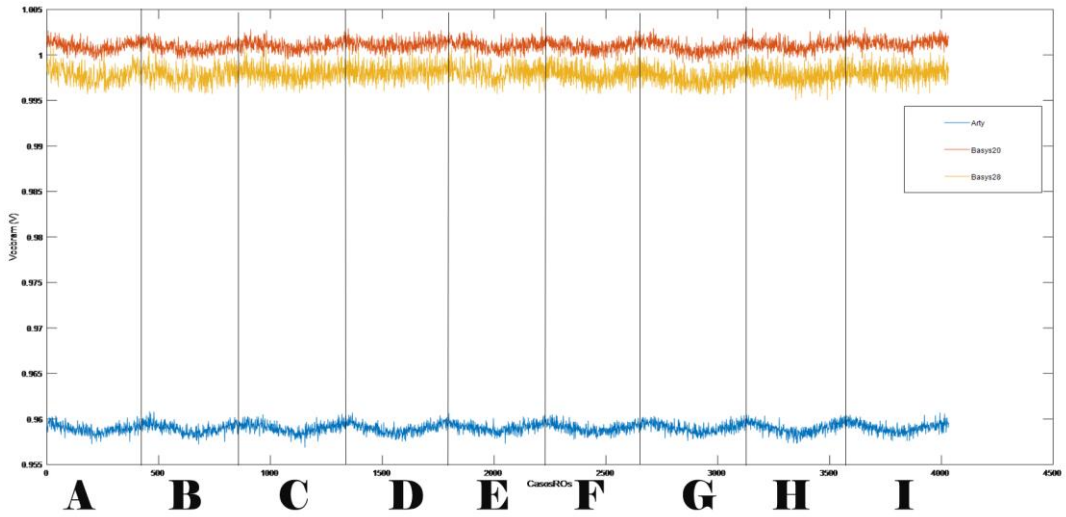


Figura 24: Gráfica de Vccbram del Experimento 1

Al enfrentar a las distintas FPGA entre sí, se puede observar que para todos los parámetros de control se alcanzan valores distintos. Así que, se puede extraer como conclusión de este experimento que existe una inter-distancia de Hamming ("Hamming Inter-Distance") cuantificable para las distintas FPGA empleadas durante los TEST de evaluación

Esto quiere decir que para el mismo estímulo de entrada y la misma configuración de PUF la respuesta de cada dispositivo es diferente y, por tanto, desvela la posibilidad de distinguirlos unos de otros debido a la huella digital que las variaciones aleatorias de los procesos de fabricación han infundido en cada uno de los equipos.

Por otra parte, desde un punto de vista puramente analítico, se puede ver cómo la temperatura de las FPGA aumenta a medida que se encienden los ROs y disminuye durante la fase de apagado. Este resultado tiene sentido y, además, muestra gráficamente cómo se ha conseguido cambiar la respuesta del sistema a través del control parcial de las condiciones de entorno de operación (siendo esta la diferencia del sistema propuesto con respecto a un Sensor PUF convencional).

Finalmente, resulta significativo que en las zonas más próximas a la parte central, esto es, las zonas D, E, I resulta más notable el impacto sobre el dispositivo. De esta manera, se puede concluir además que los sensores del XADC se encuentran próximos a la parte central del dispositivo y que, por tanto, se muestran más sensibles ante estímulos cuando estos influyen en esas zonas.

4.2. Experimento 2: Diferencias entre Temperatura media.

Tal y como se adelantó en la previa de este apartado de 'Resultados Experimentales', se han estado tomando datos de cada una de las FPGA durante 5 días.

En cada uno de esos días, durante la primera adquisición (TEST 1) se comenzó adquiriendo el valor de Temperatura del sensor térmico del XADC.

Por tanto, resulta interesante estudiar la reproducibilidad de los resultados bajo los mismos estímulos en tiempos diferentes. Es decir, durante este experimento, se probará la calidad del PUF implementado y la estabilidad de su respuesta.

Cabe mencionar que para que este experimento fuese apropiado, se debería haber usado una cámara térmica en la que las condiciones de temperatura, presión y humedad se mantuviesen constantes siempre que se tomaran los datos. Esto está relacionado con lo explicado durante el apartado 2.1.3. en el que se habla de cómo las condiciones del entorno de operación generan cierta variabilidad en la respuesta de una PUF.

Para la realización de este experimento se ha diseñado un script de Matlab el cual puede encontrarse en el 'Anexo 8'

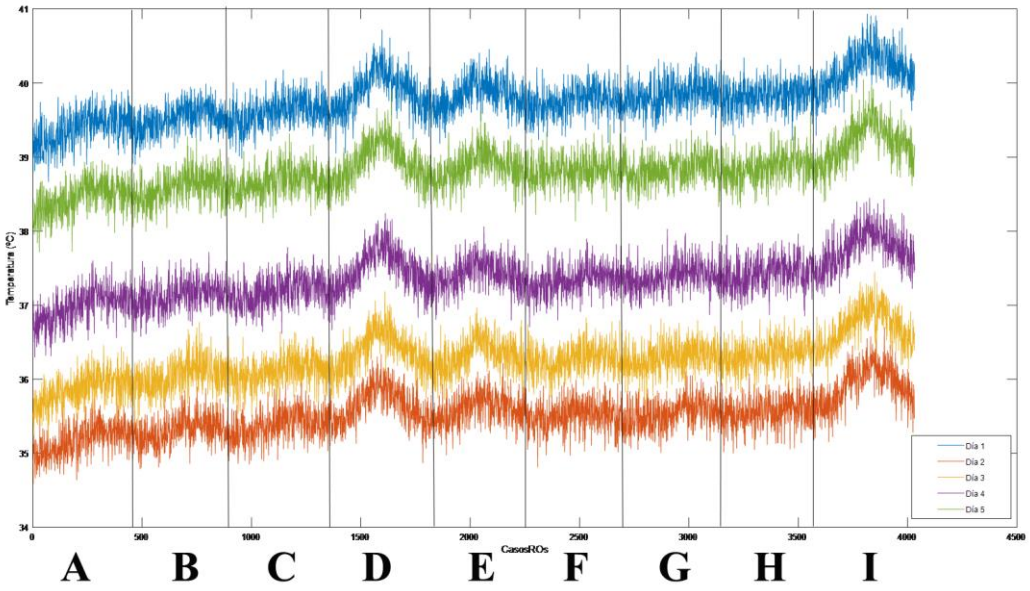


Figura 25: Gráfica de diferencia de Temperatura media del Experimento 2

En la ‘Figura 25’ se puede observar la variación de la temperatura durante cada uno de los días de toma de datos para la Arty. El resto de las FPGA siguen un comportamiento similar y, por ende, no se considera necesario la inclusión de sus gráficas dado que no aportan valor final al experimento.

En este caso, el estudio de la estabilidad de la propuesta del PUF pasa por entender que la tendencia se mantiene. Esto se debe a que no se cuenta con una cámara en la que se mantengan constantes las condiciones de entorno para estudiar la respuesta de las distintas FPGA.

Por lo tanto, es importante para analizar estos resultados saber que inevitablemente va a existir una intra-distancia de Hamming (“Hamming Intra-Distance”) por la cual las medidas del mismo sistema para el mismo estímulo varían en función de la variabilidad aleatoria de las condiciones de entorno.

Sin embargo, a pesar de que el valor medio de cada uno de los TEST de evaluación es distinto debido al cambio de temperatura ambiente para cada día de adquisición de datos, la tendencia de la forma de onda es la misma para todos y cada uno de ellos.

4.3. Experimento 3: Cambio en el orden de adquisición de datos.

Tal y como se mencionó durante la previa de este apartado (‘Resultados Experimentales’), el orden de adquisición de datos, esto es, el orden de los sensores se ha ido cambiando en cada uno de los TEST de cada uno de los días durante los que se tomaron datos.

Este experimento tiene como objetivo dilucidar si el uso previo de la FPGA para otros usos, tiene un impacto en la adquisición de futuras pruebas. Por ejemplo, ¿cómo afecta a la toma de datos de la Temperatura que la FPGA haya estado trabajando anteriormente en la adquisición de datos de los otros sensores y, por tanto, el encendido y apagado de Ring Oscillators?

Para la realización de este experimento se ha diseñado un script de Matlab el cual puede encontrarse en el ‘Anexo 9’.

- ARTY:

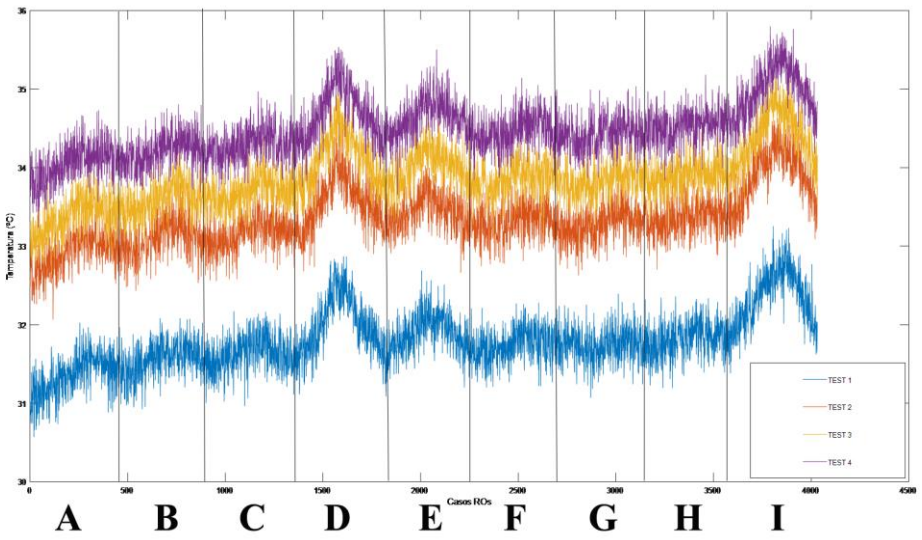


Figura 26: Gráfica de Arty del Experimento 3

- BASYS 20:

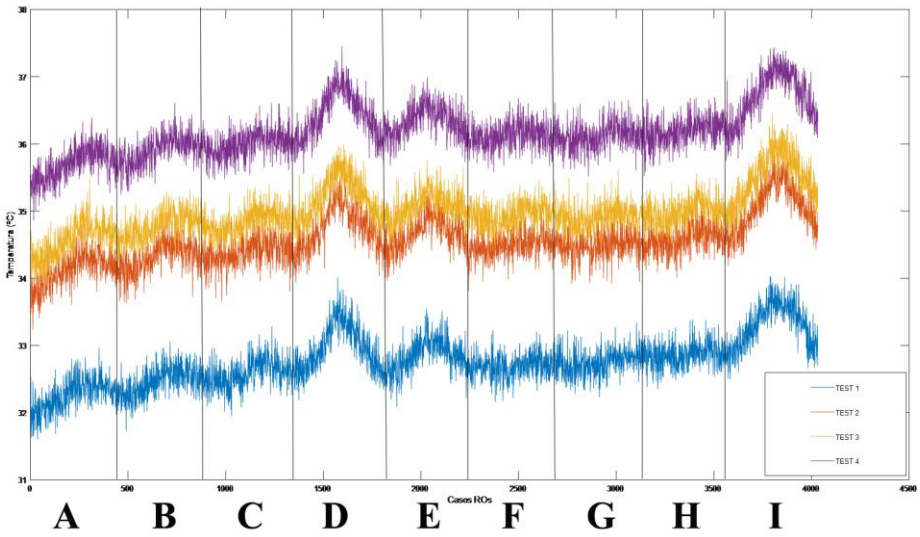


Figura 27: Gráfica de Basys20 del Experimento 3

- BASYS 28:

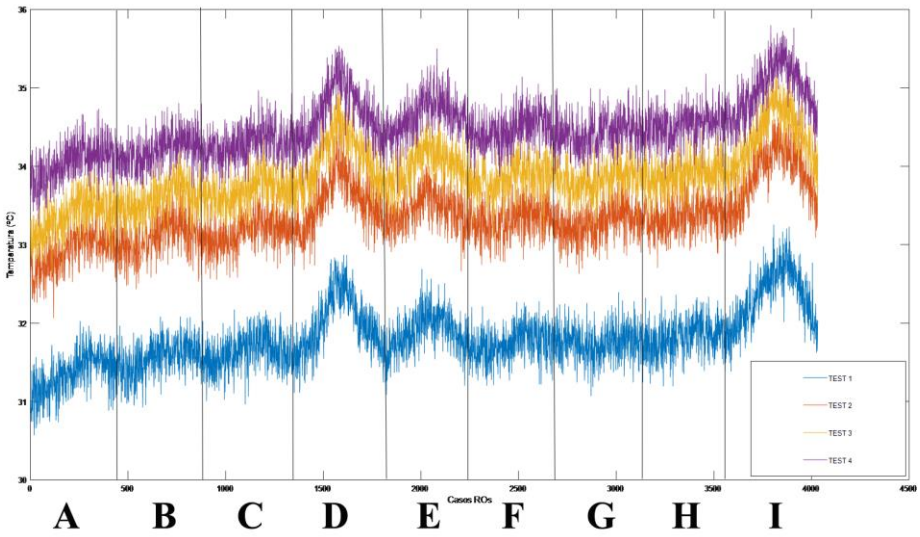


Figura 28: Gráfica de Basys28 del Experimento 3

Tras este experimento, al analizar los resultados obtenidos se puede decir que el orden de adquisición de datos tiene un impacto en la respuesta del sistema. La justificación de esta variación en los resultados viene dada por el hecho de que tras cada secuencia de encendido y apagado de ROs (uso previo de la FPGA) las condiciones de entorno cambian ligeramente para cada uno de los diferentes TEST de evaluación.

No obstante, a pesar del cambio en las condiciones de entorno de operación se puede observar como cada FPGA mantiene su tendencia en la forma de onda de la señal de respuesta. Esto es un vestigio positivo de cara a la estabilidad de la huella digital y, por ende, al uso de la configuración de PUF propuesta para la autenticación de dispositivos.

4.4. Experimento 4: Coeficiente de correlación entre distintas FPGA.

En estadística, la correlación reseña si 2 variables están relacionadas entre sí o no. Además, fija esta relación en base a:

1. Relación positiva o negativa.
2. Fuerza de dicha relación.

De esta manera, el resultado de la función de correlación varía entre -1 y +1. Indicando si es positiva o negativa por su signo, y su fuerza a través de la magnitud de su valor absoluto.

La idea de conocer este coeficiente de correlación entre las distintas FPGA viene dada por saber si es posible distinguir los distintos dispositivos por su respuesta, o si su comportamiento es tan similar que es imposible detectarlos en base a él.

Para la realización de este experimento se ha diseñado un script de Matlab el cual puede encontrarse en el 'Anexo 10'.

A través del comando 'corrcoef()' se han obtenido los siguientes resultados para cada uno de los parámetros.

- Temperatura:

Tabla 4: Coeficiente de correlación en función de Temperatura

TEMPERATURA	Arty	Basys20	Basys28
Arty	1	0.6845	0.6421
Basys20	0.6845	1	0.7202
Basys28	0.6421	0.7202	1

- Vccint:

Tabla 5: Coeficiente de correlación en función de Vccint

VCCINT	Arty	Basys20	Basys28
Arty	1	0.3828	0.2286
Basys20	0.3828	1	0.2187
Basys28	0.2286	0.2187	1

- Vccaux:

Tabla 6: Coeficiente de correlación en función de Vccaux

VCCAUX	Arty	Basys20	Basys28
Arty	1	-0.0199	0.0133
Basys20	-0.0199	1	0.0761
Basys28	0.0133	0.0761	1

- Vccbram:

Tabla 7: Coeficiente de correlación en función de Vccbram

VCCBRAM	Arty	Basys20	Basys28
Arty	1	0.2604	0.1917
Basys20	0.2604	1	0.1759
Basys28	0.1917	0.1759	1

De las tablas anteriores, al margen de que claramente:

- corrcoef(X,Y) = corrcoef(Y,X)
- corrcoef(Z,Z) = 1

De las Tablas 4, 5, 6 y 7 se puede extraer que la respuesta de cada una de las FPGA ante el mismo estímulo, con la misma configuración de PUF, es distinta y única. Es decir, se deja entrever que es posible identificar a cada uno de los dispositivos y, por ende, utilizar la configuración propuesta como una función física inclonable (PUF) con cierto grado de calidad y veracidad.

Esto se debe a que la magnitud del coeficiente de correlación para cualquier par de FPGA es distinto de 1. Por tanto, cada uno de los dispositivos deja una firma digital distinta que los diferencia y hace únicos entre sí.

Además, cabe mencionar que la temperatura es el parámetro que más correlación tiene entre los distintos dispositivos, mientras que, por ejemplo, VCCAUX tiene unos índices de correlación casi nulos.

4.5. Resumen de los resultados experimentales.

Tras todo el estudio y tratamiento de datos de la fase experimental se ha llegado a la conclusión de que a través de esta PUF, cada una de las 3 FPGA empleadas han generado una firma distinta.

En primer lugar, se puede observar gracias al Experimento 1 que las FPGA se ven afectadas no solo por las condiciones de entorno que escapan a nuestro control, sino que también se influencian por los Ring Oscillator, los cuales se podría decir que aparte de ser una condición de entorno, las hemos convertido en estímulos del sistema.

Por otra parte, se ha demostrado gracias al Experimento 4, que aunque la tendencia de las 3 FPGA se inclina a hacer lo mismo, realmente cada una de ellas genera una firma particular, distinta e inherente a cada uno de los dispositivos.

Todo lo comentado hasta ahora sobre los experimentos 1 y 4 se resume en la evidencia de que cada FPGA puede distinguirse de las demás explotando las variaciones de fabricación a través de los sensores XADC.

Además, gracias al Experimento 2 se ha visto que el índice de reproducibilidad de la respuesta a la PUF depende directamente e inevitablemente de las condiciones del entorno de operación. No obstante, la tendencia de la señal de respuesta se mantiene constante, por tanto, se consigue la estabilidad necesaria para la autenticación de los diferentes dispositivos.

Asimismo, se puede extraer del Experimento 3 que la respuesta de un sistema a una PUF viene determinado por su estado anterior. Es decir, las condiciones de entorno influyen en la reproducibilidad de la respuesta y éstas cambian debido al previo uso de los propios dispositivos (por ejemplo, si se utilizado una FPGA para otro fin anteriormente, la temperatura de la misma será mayor que si, por el contrario, ésta se encontrara en estado de reposo antes de la evaluación).

Como dato clave, el hecho de que el orden con el que se toman los datos afecte a la respuesta del PUF es algo muy positivo para la autenticación de dispositivos. Esto se debe a que se pueden generar más pares estímulo-respuesta (CRP), lo cual es crucial en PUFs.

Finalmente, se puede concluir diciendo que la PUF implementada es capaz de generar distintas firmas para cada dispositivo con un alto grado de fiabilidad.

5. Conclusiones.

5.1. Objetivos y resultados.

Como objetivo principal se planteó estudiar la posibilidad explotar las variaciones intrínsecas de los sensores (integrados en una FPGA) las cuales se deben a las imperfecciones durante los procesos de fabricación de los semiconductores. Asimismo, propuso controlar parcialmente mediante el uso de Ring Oscillators (también sujetos a las variaciones de fabricación) las condiciones del entorno de operación.

En primera instancia, fue necesaria la familiarización previa con el concepto de PUF y, sobre todo, con el módulo XADC integrado en las FPGA de Xilinx. Una vez entendido cómo funcionaba el conversor, se decidió estudiar 4 parámetros de control los cuales se creyeron significativos (Temperatura, Vccint, Vccaux, Vccbram) a la hora de intentar identificar los distintos dispositivos. Finalmente, se desarrolló el bloque VHDL a través del cual se controlaba el módulo XADC y con el cual se empezaría a intentar explotar las variaciones de fabricación de los sensores.

Por otro lado, lo que distingue a esta propuesta de PUF de las demás es que tiene 2 entradas. Esto es, aparte de la entrada tradicional de un PUF (magnitud física), aquí se cuenta con los Ring Oscillator los cuales a la vez son parte de las condiciones de entorno ya que su funcionamiento implica cambios en la temperatura del sistema. Se han implementado 9 zonas con 224 instanciaciones de ROs cada una de ellas, haciendo un total de 2016 ROs distribuidos por toda la FPGA (todo esto a través de una XDC macro).

Una vez completada la configuración del PUF, se ha procedido a estudiarlo a través de 4 experimentos distintos de cara a comprobar la idoneidad del sistema propuesto. Esto se ha conseguido transmitiendo los datos vía UART a un ordenador.

Finalmente, tras analizar los datos obtenidos, se puede concluir que el sistema propuesto puede ser usado para la autenticación de dispositivos. Tal y como se ha resumido en el apartado 4.5., existe cierta correlación entre las distintas FPGA para cada uno de los parámetros de control, pero al final, cada una responde de forma distinta. Asimismo, se puede ver cómo se ha podido influir sobre el sistema controlando parcialmente las condiciones de operación (Ring Oscillators), siendo éstas incontrolables e impuestas por el entorno en las configuraciones convencionales de PUF. Por lo tanto, se ha cumplido de manera satisfactoria con el objetivo del proyecto.

5.2. Líneas futuras.

Este trabajo ha contribuido a despejar algunas incógnitas sobre el tema tratado, como la utilidad de los sensores XADC para la autenticación de FPGA mediante un PUF controlando parcialmente las condiciones de operación (Ring Oscillator), pero a la vez genera nuevas preguntas y propuestas de mejora.

Dichas propuestas de mejora o líneas de investigación futuras se resumen en:

- Repetición de los TEST de evaluación bajo un entorno de mínimo cambio como una cámara térmica en la que se tengan controlados parámetros tales como la temperatura, la presión e incluso la humedad.

- Aumentar el número de FPGA a identificar para comprobar que el sistema sea capaz de identificar cada una de ellas.

- Ampliar el método de autenticación propuesto a otras tecnologías de FPGA para validar la fiabilidad del sistema propuesto.

- Probar la configuración de PUF propuesto en otro tipo de dispositivos y tecnologías como, por ejemplo, microprocesadores.

- Crear un protocolo de autenticación machine-learning (redes neuronales).

6. Planificación y presupuesto.

6.1. Planificación de trabajo.

- **Documentación:** estudio inicial del arte y familiarización con los conceptos básicos sobre el tema del trabajo.

- **Diseño del sistema:** diseño de un sistema (programado en lenguaje VHDL) que permita obtener información de los sensores, y escupa los datos a través de puerto serie de la FPGA. Además, en este apartado se incluye el proceso de creación de una XDC macro, la instanciación de los Ring Oscillator, y la final implementación de un RO-PUF.

- **Test sobre FPGA:** realización de todas las pruebas necesarias para la obtención de los datos a analizar.

- **Tratamiento de datos:** acondicionamiento de los datos obtenidos para sacar conclusiones oportunas de dichos resultados. Este apartado incluye el código C++ usado para el acondicionamiento de los archivos .txt, los scripts de Matlab empleados para graficar, etc.

- **Redacción de la memoria:** elaboración del documento descriptivo del trabajo.

Tabla 8: Planificación de Trabajo.

PLANIFICACIÓN DE TRABAJO		
Tarea	**Duración (horas)**	**Avance (%)**
Documentación	**60**	**20**
Diseño del sistema	**126**	**42**
Test sobre FPGA	**5**	**1.67**
Tratamiento de datos	**45**	**15**
Redacción de la memoria	**64**	**21.33**
TOTAL	**300**	**100**

6.2. Presupuesto.

6.2.1. Capítulo I: Materiales y recursos.

En este capítulo no se incluyen los precios de herramientas software utilizados como Vivado 2018.3 y DevC++. El motivo es que se han usado versiones gratuitas, y por tanto, no suman valor final al presupuesto total.

Tabla 9: Presupuesto. Capítulo I: Materiales y recursos.

Unidad	Descripción	Precio Unitario	Medición	Precio final
Ud.	**ASUS ROG GL553VD.**			
	Ordenador portátil para el uso de las herramientas software requeridas para la implementación de la función física inclonable en FPGA y el tratamiento de los datos obtenidos, así como la elaboración de la memoria descriptiva.			
		799.00	1	
				799.00
Ud.	**Basys 3 Artix-7 FPGA Trainer Board.**			
	Placa de prototipado versátil de nivel básico para propósitos académicos perfecta para principiantes y estudiantes que dan sus primeros pasos con tecnología FPGA. Part: xc7a35tcpg236-1.			
		131.82	2	
				263.64
Ud.	**Arty Artix-7 FPGA Evaluation Kit.**			
	Placa de prototipado diseñada por Xilinx. Destaca por la gran flexibilidad para trabajar en distintas aplicaciones y por su uso intuitivo y sencillo. Part: xc7a35tcsg324-1-			
		87.59	1	
				87.59

Ud. **Cable USB.**

Cable USB 2.0, 1.5m, Negro, USB A macho a Mini USB macho tipo B. Permite la adquisición de datos de la FPGA por puerto serie.

2.50 1

2.50

Ud. **Software Matlab**

Licencia académica.
Suscripción anual.

250 1

250

TOTAL CAPÍTULO I: Materiales y recursos.....1402.73 EUROS

6.2.2. Capítulo II: Personal.

- **Ingeniero Junior:** estudiante de último año o recién titulado responsable de la elaboración del presente proyecto.
- **Ingeniero Senior:** jefe de proyecto que supervisa el trabajo realizado y guía al ingeniero junior durante la realización del mismo.

Tabla 10: Presupuesto. Capítulo II: Personal

Presupuesto. Capítulo II: Personal			
Personal	**Salario (€/hora)**	**Dedicación (horas)**	**Coste (€)**
Ingeniero Junior	**11.50**	**300**	**3450.00**
Jefe de Proyecto	**15.00**	**50**	**750.00**
		TOTAL	**4200.00**

TOTAL CAPÍTULO II: Personal.....4200.00 EUROS

6.2.3. Resumen de presupuesto.

Tabla 11: Resumen del presupuesto.

Capítulo	Resumen	Importe	%
I	Materiales y recursos	1402.73	25.04
II	Personal	4200.00	74.96
		5602.73	100

Asciende el presupuesto general la cantidad de CINCO MIL SEISCIENTOS DOS CON SETENTA Y TRES EUROS (5602.73 €).

Bibliografía

[1] I. Mártil, «El circuito integrado: la tecnología que cambió nuestra vida.,» 15 4 2016. [En línea]. Available: https://blogs.publico.es/ignacio-martil/2016/04/15/el-circuito-integrado-la-tecnologia-que-cambio-nuestra-vida/?doing_wp_cron=1560510073.7767899036407470703125.

[2] J. L. Sagarduy, «Falsificación y piratería industrial. El caso de China.,» 15 3 2012. [En línea]. Available: http://www.madrimasd.org/informacionIdi/analisis/opinion/opinion.asp?id=51930.

[3] G. Wood, «Costly counterfeit electronic components in the supply chain can also be a safety concern,» 06 Enero 2016. [En línea]. Available: https://ihsmarkit.com/research-analysis/costly-counterfeit-electronic-components-in-the-supply-chain-can-also-be-a-safety-concern.html.

[4] J. A. Roy, F. Koushanfar y I. L. Markov, «EPIC: Ending Piracy of Integrated Circuits,» 2008.

[5] C. Mesaritakis, M. Akriotou, A. Kapsalis, E. Grivas, C. Chaintoutis, T. Nikas y D. Syvridis, «www.nature.com/scientificreports,» 25 Junio 2018. [En línea]. Available: https://www.nature.com/articles/s41598-018-28008-6.

[6] R. Maes y I. Verbauwhede, Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions., Springer-Verlag Berlin Heidelberg, 2010.

[7] B. Halak, M. Zwolinski y M. S. Mispan, «Overview of PUF-based hardware security solutions for the internet of things,» 2016.

[8] G. E. Suh y S. Devadas, «Physical Unclonable Functions for Device Authentication and Secret Key Generation,» 2007.

[9] R. Maes, A. Van Herrewege y I. Verbauwhede, «PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator».

[10] M. Ayat, R. Ebrahimi Atani y S. Mirzakuchaki, On Design of PUF-Based Random Number Generators, vol. 3, 2011.

[11] F. Amsaad, M. Niamat, A. Dawoud y S. Kose, Reliable Delay Based Algorithm to Boost PUF Security Against Modeling Attacks, 2018.

[12] Intrinsic ID, «Instinsic ID SRAM PUF Technology,» [En línea]. Available: https://www.intrinsic-id.com/sram-puf-technology/.

[13] H. Ma, Y. Gao, O. Kavehei y D. C. Ranasinghe, «A PUF Sensor: Securing Physical Measurements,» 2017.

[14] K. Rosenfeld, E. Gavas y R. Karri, Sensor physical unclonable functions., 2010.

[15] S. Chen, B. Li y Y. Cao, «Intrinsic Physical Unclonable Function (PUF) Sensors in Commodity Devices,» 2019.

[16] XilinxInc, «Vivado Design Suite User Guide: Using Constraints,» 20 3 2013. [En línea]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_1/ug903-vivado-using-constraints.pdf.

[17] XilinxInc, «Vivado XDC Macro Creation [Video File],» 1 8 2013. [En línea]. Available: https://www.youtube.com/watch?v=hJ1LaSKYYow&t=214s.

[18] XilinxInc, «7 Series FPGAs and Zynq-7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Ditial Converter. User Guide.,» 23 7 2018. [En línea]. Available: https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf.

Anexo 1: Código VHDL. Ring Oscillator.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library unisim;
use unisim.vcomponents.all;
--
----------------------------------------------------------------------
--
-- Main Entity for ring_osc
--
entity ring_osc is
port(   ro_en : in std_logic;
        pulse_out: out std_logic);
end ring_osc;

 architecture low_level_definition of ring_osc is


--signal gate_out : std_logic_vector(5 downto 0) := (others => '0
signal gate_out_0 : std_logic;
signal gate_out_1 : std_logic ;
signal gate_out_2 : std_logic;
signal gate_out_3 : std_logic ;


attribute KEEP : string;
attribute KEEP of gate_out_0 : signal is "true";
attribute KEEP of gate_out_1 : signal is "true";
attribute KEEP of gate_out_2 : signal is "true";
attribute KEEP of gate_out_3 : signal is "true";


begin
process(gate_out_0, gate_out_1, gate_out_2, gate_out_3, ro_en)
begin

    gate_out_0 <= ro_en and gate_out_3;
    gate_out_1 <= not(gate_out_0);
    gate_out_2 <= not(gate_out_1);
    gate_out_3 <= not(gate_out_2);

    pulse_out<=gate_out_3;
end process;



end low_level_definition;
```

Anexo 2: Código VHDL. UART.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity UART_TX is
  generic (
    g_CLKS_PER_BIT : integer := 869     -- 100MHz / 115200 baud = 868.0505
    );
  port (
    i_Clk       : in  std_logic;
    i_TX_DV     : in  std_logic;
    i_TX_Byte   : in  std_logic_vector(7 downto 0);
    o_TX_Active : out std_logic;
    o_TX_Serial : out std_logic;
    o_TX_Done   : out std_logic
    );
end UART_TX;



architecture RTL of UART_TX is

  type t_SM_Main is (s_Idle, s_TX_Start_Bit, s_TX_Data_Bits,
                     s_TX_Stop_Bit, s_Cleanup);
  signal r_SM_Main : t_SM_Main := s_Idle;

  signal r_Clk_Count : integer range 0 to g_CLKS_PER_BIT-1 := 0;
  signal r_Bit_Index : integer range 0 to 7 := 0;  -- 8 Bits Total
  signal r_TX_Data   : std_logic_vector(7 downto 0) := (others => '0');
  signal r_TX_Done   : std_logic := '0';

begin


  p_UART_TX : process (i_Clk)
  begin
    if rising_edge(i_Clk) then

      case r_SM_Main is
```

```
      when s_Idle =>
        o_TX_Active <= '0';
        o_TX_Serial <= '1';         -- Drive Line High for Idle
        r_TX_Done   <= '0';
        r_Clk_Count <= 0;
        r_Bit_Index <= 0;

        if i_TX_DV = '1' then
          r_TX_Data <= i_TX_Byte;
          r_SM_Main <= s_TX_Start_Bit;
        else
          r_SM_Main <= s_Idle;
        end if;


      -- Send out Start Bit. Start bit = 0
      when s_TX_Start_Bit =>
        o_TX_Active <= '1';
        o_TX_Serial <= '0';

        -- Wait g_CLKS_PER_BIT-1 clock cycles for start bit to finish
        if r_Clk_Count < g_CLKS_PER_BIT-1 then
          r_Clk_Count <= r_Clk_Count + 1;
          r_SM_Main   <= s_TX_Start_Bit;
        else
          r_Clk_Count <= 0;
          r_SM_Main   <= s_TX_Data_Bits;
        end if;


      -- Wait g_CLKS_PER_BIT-1 clock cycles for data bits to finish
      when s_TX_Data_Bits =>
        o_TX_Serial <= r_TX_Data(r_Bit_Index);

        if r_Clk_Count < g_CLKS_PER_BIT-1 then
          r_Clk_Count <= r_Clk_Count + 1;
          r_SM_Main   <= s_TX_Data_Bits;
        else
          r_Clk_Count <= 0;

          -- Check if we have sent out all bits
          if r_Bit_Index < 7 then
            r_Bit_Index <= r_Bit_Index + 1;
            r_SM_Main   <= s_TX_Data_Bits;
          else
            r_Bit_Index <= 0;
            r_SM_Main   <= s_TX_Stop_Bit;
          end if;
        end if;


      -- Send out Stop bit.  Stop bit = 1
      when s_TX_Stop_Bit =>
        o_TX_Serial <= '1';

        -- Wait g_CLKS_PER_BIT-1 clock cycles for Stop bit to finish
        if r_Clk_Count < g_CLKS_PER_BIT-1 then
          r_Clk_Count <= r_Clk_Count + 1;
          r_SM_Main   <= s_TX_Stop_Bit;
        else
          r_TX_Done   <= '1';
          r_Clk_Count <= 0;
          r_SM_Main   <= s_Cleanup;
        end if;
```

```
        -- Stay here 1 clock
        when s_Cleanup =>
          o_TX_Active <= '0';
          r_TX_Done   <= '1';
          r_SM_Main   <= s_Idle;


        when others =>
          r_SM_Main <= s_Idle;

      end case;
    end if;
  end process p_UART_TX;

  o_TX_Done <= r_TX_Done;

end RTL;
```

Anexo 3: Código VHDL. XADC.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity xadc is
port(
        clk: in STD_LOGIC;
        reset: in STD_LOGIC;
        switch: in STD_LOGIC_VECTOR(1 downto 0);
      --vp: in STD_LOGIC;
      --vn: in STD_LOGIC;
        end_conv: out STD_LOGIC;
        measured: out STD_LOGIC_VECTOR(15 downto 0));
end xadc;


architecture Behavioral of xadc is

signal ready, enable: STD_LOGIC;
--signal bad_address: STD_LOGIC:='0';
signal readyaux: STD_LOGIC:='0';
signal data: STD_LOGIC_VECTOR(15 downto 0);
signal Address_in: STD_LOGIC_VECTOR(6 downto 0);
signal busy_out, eos_out, alarm_out: std_logic;
signal channel_out: std_logic_vector(4 downto 0);
signal vp, vn: std_logic:='0';
```

```
component xadc_wiz_0 is
   port(
 daddr_in        : in  STD_LOGIC_VECTOR (6 downto 0);     -- Address bus for the dynamic reconfiguration port
 den_in          : in  STD_LOGIC;                         -- Enable Signal for the dynamic reconfiguration port
 di_in           : in  STD_LOGIC_VECTOR (15 downto 0);    -- Input data bus for the dynamic reconfiguration port
 dwe_in          : in  STD_LOGIC;                         -- Write Enable for the dynamic reconfiguration port
 do_out          : out  STD_LOGIC_VECTOR (15 downto 0);   -- Output data bus for dynamic reconfiguration port
 drdy_out        : out  STD_LOGIC;                        -- Data ready signal for the dynamic reconfiguration port
 dclk_in         : in  STD_LOGIC;                         -- Clock input for the dynamic reconfiguration port
 reset_in        : in  STD_LOGIC;                         -- Reset signal for the System Monitor control logic
 busy_out        : out  STD_LOGIC;                        -- ADC Busy signal
 channel_out     : out  STD_LOGIC_VECTOR (4 downto 0);    -- Channel Selection Outputs
 eoc_out         : out  STD_LOGIC;                        -- End of Conversion Signal
 eos_out         : out  STD_LOGIC;                        -- End of Sequence Signal
 alarm_out       : out STD_LOGIC;                         -- OR'ed output of all the Alarms
 vp_in           : in  STD_LOGIC;                         -- Dedicated Analog Input Pair
 vn_in           : in  STD_LOGIC
);
end component;




begin
xadc: xadc_wiz_0 port map(  daddr_in => Address_in,
                            den_in => enable,
                            di_in => (others=>'0'),
                            dwe_in => '0',
                            do_out => data,
                            drdy_out => ready,
                            dclk_in => clk,
                            reset_in => reset,
                            busy_out => busy_out,
                            channel_out => channel_out,
                            eoc_out => enable,
                            eos_out => eos_out,
                            alarm_out => alarm_out,
                            vp_in => vp,
                            vn_in => vn);

end_conv <= enable;
process(clk, reset)
begin
if reset='1' then
    readyaux<='0';
    measured<= (others=>'0');
elsif rising_edge(clk) then
    readyaux<=ready;
    if ready='1' AND readyaux='0'then
        measured <= data;
    end if;
end if;
end process;

process(clk)
begin
if rising_edge(clk) then
    if ready='0' and readyaux='1' then
--        bad_address<='0';
        case switch is
            when "00" => Address_in <= std_logic_vector(to_unsigned(0, 7));     -- temperatura 00
            when "01" => Address_in <= std_logic_vector(to_unsigned(1, 7));     -- vccint 01
            when "10" => Address_in <= std_logic_vector(to_unsigned(2, 7));     -- vccaux 02
            when "11" => Address_in <= std_logic_vector(to_unsigned(6, 7));     -- vccbram 06
            when others=>   Address_in <= (others=>'0');
                          --  bad_address<='1';
        end case;
    end if;
end if;
end process;



end Behavioral;
```

Anexo 4: Script Matlab. Archivo .xdc con el que se posicionan los ROs en la FPGA.

```
clear; clc;
int8 a=0;
int8 b=0;
int8 c=0;
int8 d=0;

fileID = fopen('place_ro.txt', 'w');

for a=0:1:8
    switch a
        case 0
            for b=0:1:13
                for c=0:1:15
                        d= c + 16*b;
                        % set_property LOC SLICE_X0Y149 [get_cells {ring_gen[0].roA/gate_out_0_inferred_i_1}]
                        fprintf(fileID, 'set_property LOC SLICE_X%dY%d [get_cells {ring_gen[%d].roA/gate_out_0_inferred_i_1}]\n',b, (149-c),d);
                end
            end
            fprintf(fileID,'\n\n');
        case 1
            for c=0:1:15
                for b=0:1:13
                        d= b + 14*c;
                        fprintf(fileID, 'set_property LOC SLICE_X%dY%d [get_cells {ring_gen[%d].roB/gate_out_0_inferred_i_1}]\n',(28+b), (149-c),d);
                end
            end
            fprintf(fileID,'\n\n');
        case 2
            for b=0:1:13
                for c=0:1:15
                        d= c + 16*b;
                        fprintf(fileID, 'set_property LOC SLICE_X%dY%d [get_cells {ring_gen[%d].roC/gate_out_0_inferred_i_1}]\n',(57-b), (149-c),d);

                end
            end
            fprintf(fileID,'\n\n');

        case 3
            for b=0:1:13
                for c=0:1:15
                        d= c + 16*b;
                        fprintf(fileID, 'set_property LOC SLICE_X%dY%d [get_cells {ring_gen[%d].roD/gate_out_0_inferred_i_1}]\n',b, (c+67),d);

                end
            end
            fprintf(fileID,'\n\n');
        case 4
            for b=0:1:13
                for c=0:1:15
                        d= c + 16*b;
                        fprintf(fileID, 'set_property LOC SLICE_X%dY%d [get_cells {ring_gen[%d].roE/gate_out_0_inferred_i_1}]\n',(57-b), (c+67),d);
                end

            end
            fprintf(fileID,'\n\n');
        case 5
            for b=0:1:13
                for c=0:1:15
                        d= c + 16*b;
                        fprintf(fileID, 'set_property LOC SLICE_X%dY%d [get_cells {ring_gen[%d].roF/gate_out_0_inferred_i_1}]\n',b, c,d);

                end
            end
            fprintf(fileID,'\n\n');
        case 6
            for c=0:1:15
                for b=0:1:13
                        d= b + 14*c;
                        fprintf(fileID, 'set_property LOC SLICE_X%dY%d [get_cells {ring_gen[%d].roG/gate_out_0_inferred_i_1}]\n',(28+b), c,d);

                end
            end
            fprintf(fileID,'\n\n');

        case 7
            for b=0:1:13
                for c=0:1:15
                        d= c + 16*b;
                        fprintf(fileID, 'set_property LOC SLICE_X%dY%d [get_cells {ring_gen[%d].roH/gate_out_0_inferred_i_1}]\n',(57-b), c,d);

                end
            end
            fprintf(fileID,'\n\n');
        case 8
            for b=0:1:13
                for c=0:1:15
                        d= c + 16*b;
                        fprintf(fileID, 'set_property LOC SLICE_X%dY%d [get_cells {ring_gen[%d].roI/gate_out_0_inferred_i_1}]\n',(b+28),(c+67),d);
                end

            end
            fprintf(fileID,'\n\n');
    end
end
fclose(fileID);
```

Anexo 5: Código C++. Separación de cada una de las medidas obtenidas en 6 dígitos.

```
#include <iostream>
#include <stdlib.h>
#include <fstream>
#include <string>
using namespace std;

string texto;
int count = 0;
int jump = 0;
ofstream archivo;

void leer();
void escribir();
int main(){

    leer();
    escribir();
    for (int i=0; i<texto.length()+6*2*3*224; i=i+6)
    {
        if ((count+1)%16 == 0){
            jump = jump + 2;
            count=0;
        }else{

        archivo<<texto[i-jump]<<texto[i+1-jump]<<texto[i+2-jump]<<texto[i+3-jump]<<texto[i+4-jump]<<texto[i+5-jump]<<endl;
        count = count + 1;
        }
    }
    archivo.close();
    system("pause");
    return 0;
}


void leer(){
    ifstream archivo1;
    archivo1.open("Basys28_vccaux", ios::in);
    if (archivo1.fail()){
        cout<<"Error en lectura de archivo"<<endl<<endl;
        exit(1);
    }

    while(!archivo1.eof()){
        getline(archivo1, texto);
        archivo1.close();
    }
}
void escribir(){

    archivo.open("Basys28_vccaux_6", ios::out);
    if(archivo.fail()){
        cout<<"Error creación de archivo"<<endl<<endl;
        exit(1);
    }
}
```

Anexo 6: Código C++. Separación de cada medida en 3 archivos txt.

```
#include <iostream>
#include <stdlib.h>
#include <fstream>
#include <string>
using namespace std;


string texto;
int count = 0;
int jump = 0;

ofstream archivow_medida, archivow_zona, archivow_ro;
ifstream archivor;


int main(){

    archivor.open("Basys28_vccaux_6", ios::in);
    archivow_medida.open("medida", ios::out);
    archivow_ro.open("ro", ios::out);
    archivow_zona.open("zona", ios::out);

    while(!archivor.eof()){
        getline(archivor, texto);
        archivow_medida << texto[1] << texto[2] << texto[3] << endl;
        archivow_ro << texto[4] << texto[5] <<endl;
        archivo_zona << texto[0]<<endl;
    }

    archivor.close();
    archivow_medida.close();
    archivow_ro.close();
    archivow_zona.close();

    system("pause");
    return 0;
}
```

Anexo 7: Script Matlab. Experimento 1. Comparativo entre las distintas FPGA.

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    C:\Users\yo\Desktop\ADC_PUF\Data_PUF\Data_PUF\Basys_20\Basys_20_T\Basys20_T_20mayo_1139_medida.txt
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

% Auto-generated by MATLAB on 2019/05/23 18:02:12

%% Initialize variables.
%filename = 'C:\Users\yo\Desktop\ADC_PUF\Data_PUF\Data_PUF\Basys_20\Basys_20_T\Basys20_T_20mayo_1139_medida.txt';
filename = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia1\TEST1\Arty_vccaux\medida';
filename1 = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia1\TEST1\Basys20_vccaux\medida';
filename2 = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia1\TEST1\Basys28_vccaux\medida';
delimiter = {''};

%% Format for each line of text:
%   column1: text (%s)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');
fileID1 = fopen(filename1,'r');
fileID2 = fopen(filename2,'r');

%% Read columns of data according to the format.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);
dataArray1 = textscan(fileID1, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);
dataArray2 = textscan(fileID2, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);

%% Close the text file.
fclose(fileID);
fclose(fileID1);
fclose(fileID2);


%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Create output variable
ArtyTempMedida = table(dataArray{1:end-1}, 'VariableNames', {'B0'});
Basys20Medida = table(dataArray1{1:end-1}, 'VariableNames', {'B1'});
Basys28Medida = table(dataArray2{1:end-1}, 'VariableNames', {'B2'});

%% Clear temporary variables
clearvars filename delimiter formatSpec fileID dataArray ans;
clearvars filename1 delimiter formatSpec fileID1 dataArray1 ans;
clearvars filename2 delimiter formatSpec fileID2 dataArray2 ans;

%%  Temperature conversion

% Arty_converted=hex2dec(ArtyTempMedida.B0)*503.975/4096-273.15;
% Basys20_converted=hex2dec(Basys20Medida.B1)*503.975/4096-273.15;
% Basys28_converted=hex2dec(Basys28Medida.B2)*503.975/4096-273.15;

%% Voltage conversion


Arty_converted=hex2dec(ArtyTempMedida.B0)*3/4096;
Basys20_converted=hex2dec(Basys20Medida.B1)*3/4096;
Basys28_converted=hex2dec(Basys28Medida.B2)*3/4096;

%% Mean data

  for j=1:448*9
        if j==1
            media_Arty(j,1)=mean(Arty_converted(4:j*15));
            media_Basys20(j,1)=mean(Basys20_converted(4:j*15));
            media_Basys28(j,1)=mean(Basys28_converted(4:j*15));
        else
            media_Arty(j,1)=mean(Arty_converted((15*j-14):j*15));
            media_Basys20(j,1)=mean(Basys20_converted((15*j-14):j*15));
            media_Basys28(j,1)=mean(Basys28_converted((15*j-14):j*15));
        end
  end

Plot_exp1 = horzcat(media_Arty,media_Basys20, media_Basys28);
save  VccauxMedida.mat;
```

Anexo 8: Script Matlab. Experimento 2. Diferencias entre temperatura media.

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    C:\Users\yo\Desktop\ADC_PUF\Data_PUF\Data_PUF\Basys_20\Basys_20_T\Basys20_T_20mayo_1139_medida.txt
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

% Auto-generated by MATLAB on 2019/05/23 18:02:12

%% Initialize variables.
%filename = 'C:\Users\yo\Desktop\ADC_PUF\Data_PUF\Data_PUF\Basys_20\Basys_20_T\Basys20_T_20mayo_1139_medida.txt';
filename = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia1\TEST2\Basys28_vccaux\medida';
filename1 = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia2\TEST2\Basys28_vccaux\medida';
filename2 = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia3\TEST2\Basys28_vccaux\medida';
filename3 = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia4\TEST2\Basys28_vccaux\medida';
filename4 = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia5\TEST2\Basys28_vccaux\medida';
delimiter = {''};

%% Format for each line of text:
%   column1: text (%s)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');
fileID1 = fopen(filename1,'r');
fileID2 = fopen(filename2,'r');
fileID3 = fopen(filename3,'r');
fileID4 = fopen(filename4,'r');

%% Read columns of data according to the format.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);
dataArray1 = textscan(fileID1, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);
dataArray2 = textscan(fileID2, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);
dataArray3 = textscan(fileID3, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);
dataArray4 = textscan(fileID4, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);

%% Close the text file.
fclose(fileID);
fclose(fileID1);
fclose(fileID2);
fclose(fileID3);
fclose(fileID4);
```

```
%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Create output variable
Dia1_Medida = table(dataArray{1:end-1}, 'VariableNames', {'B0'});
Dia2_Medida = table(dataArray1{1:end-1}, 'VariableNames', {'B1'});
Dia3_Medida = table(dataArray2{1:end-1}, 'VariableNames', {'B2'});
Dia4_Medida = table(dataArray3{1:end-1}, 'VariableNames', {'B3'});
Dia5_Medida = table(dataArray4{1:end-1}, 'VariableNames', {'B4'});

%% Clear temporary variables
clearvars filename delimiter formatSpec fileID dataArray ans;
clearvars filename1 delimiter formatSpec fileID1 dataArray1 ans;
clearvars filename2 delimiter formatSpec fileID2 dataArray2 ans;
clearvars filename3 delimiter formatSpec fileID3 dataArray3 ans;
clearvars filename4 delimiter formatSpec fileID4 dataArray4 ans;

%%  Temperature conversion

%Dia1_converted=hex2dec(Dia1_Medida.B0)*503.975/4096-273.15;
%Dia2_converted=hex2dec(Dia2_Medida.B1)*503.975/4096-273.15;
%Dia3_converted=hex2dec(Dia3_Medida.B2)*503.975/4096-273.15;
%Dia4_converted=hex2dec(Dia4_Medida.B3)*503.975/4096-273.15;
%Dia5_converted=hex2dec(Dia5_Medida.B4)*503.975/4096-273.15;

%% Voltage conversion

Dia1_converted=hex2dec(Dia1_Medida.B0)*3/4096;
Dia2_converted=hex2dec(Dia2_Medida.B1)*3/4096;
Dia3_converted=hex2dec(Dia3_Medida.B2)*3/4096;
Dia4_converted=hex2dec(Dia4_Medida.B3)*3/4096;
Dia5_converted=hex2dec(Dia5_Medida.B4)*3/4096;


%% Mean data

  for j=1:448*9
        if j==1
            media_Dia1(j,1)=mean(Dia1_converted(4:j*15));
            media_Dia2(j,1)=mean(Dia2_converted(4:j*15));
            media_Dia3(j,1)=mean(Dia3_converted(4:j*15));
            media_Dia4(j,1)=mean(Dia4_converted(4:j*15));
            media_Dia5(j,1)=mean(Dia5_converted(4:j*15));
        else
            media_Dia1(j,1)=mean(Dia1_converted((15*j-14):j*15));
            media_Dia2(j,1)=mean(Dia2_converted((15*j-14):j*15));
            media_Dia3(j,1)=mean(Dia3_converted((15*j-14):j*15));
            media_Dia4(j,1)=mean(Dia4_converted((15*j-14):j*15));
            media_Dia5(j,1)=mean(Dia5_converted((15*j-14):j*15));
        end
  end

Plot_dias = horzcat(media_Dia1, media_Dia2, media_Dia3, media_Dia4, media_Dia5);

Diff12 = media_Dia1 - media_Dia2;
Diff13 = media_Dia1 - media_Dia3;
Diff14 = media_Dia1 - media_Dia4;
Diff15 = media_Dia1 - media_Dia5;

Plot_diff = horzcat(Diff12, Diff13, Diff14, Diff15);
save  Basys28_vccaux_exp2.mat;
```

Anexo 9: Script Matlab. Experimento 3. Cambio en el orden de adquisición de datos.

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    C:\Users\yo\Desktop\ADC_PUF\Data_PUF\Data_PUF\Basys_20\Basys_20_T\Basys20_T_20mayo_1139_medida.txt
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

% Auto-generated by MATLAB on 2019/05/23 18:02:12

%% Initialize variables.
%filename = 'C:\Users\yo\Desktop\ADC_PUF\Data_PUF\Data_PUF\Basys_20\Basys_20_T\Basys20_T_20mayo_1139_medida.txt';
filename = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia2\TEST1\Basys28_vccaux\medida';
filename1 = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia2\TEST2\Basys28_vccaux\medida';
filename2 = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia2\TEST3\Basys28_vccaux\medida';
filename3 = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia2\TEST4\Basys28_vccaux\medida';
delimiter = {''};

%% Format for each line of text:
%   column1: text (%s)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');
fileID1 = fopen(filename1,'r');
fileID2 = fopen(filename2,'r');
fileID3 = fopen(filename3,'r');

%% Read columns of data according to the format.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);
dataArray1 = textscan(fileID1, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);
dataArray2 = textscan(fileID2, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);
dataArray3 = textscan(fileID3, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);

%% Close the text file.
fclose(fileID);
fclose(fileID1);
fclose(fileID2);
fclose(fileID3);


%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Create output variable
Medida_1 = table(dataArray{1:end-1}, 'VariableNames', {'B0'});
Medida_2 = table(dataArray1{1:end-1}, 'VariableNames', {'B1'});
Medida_3 = table(dataArray2{1:end-1}, 'VariableNames', {'B2'});
Medida_4 = table(dataArray3{1:end-1}, 'VariableNames', {'B3'});

%% Clear temporary variables
clearvars filename delimiter formatSpec fileID dataArray ans;
clearvars filename1 delimiter formatSpec fileID1 dataArray1 ans;
clearvars filename2 delimiter formatSpec fileID2 dataArray2 ans;
clearvars filename3 delimiter formatSpec fileID3 dataArray3 ans;

%%  Temperature conversion

% Medida_1_converted=hex2dec(Medida_1.B0)*503.975/4096-273.15;
% Medida_2_converted=hex2dec(Medida_2.B1)*503.975/4096-273.15;
% Medida_3_converted=hex2dec(Medida_3.B2)*503.975/4096-273.15;
% Medida_4_converted=hex2dec(Medida_4.B3)*503.975/4096-273.15;

%% Voltage conversion

Medida_1_converted=hex2dec(Medida_1.B0)*3/4096;
Medida_2_converted=hex2dec(Medida_2.B1)*3/4096;
Medida_3_converted=hex2dec(Medida_3.B2)*3/4096;
Medida_4_converted=hex2dec(Medida_4.B3)*3/4096;
```

```
%% Mean data

  for j=1:448*9
        if j==1
            media_Medida_1(j,1)=mean(Medida_1_converted(4:j*15));
            media_Medida_2(j,1)=mean(Medida_2_converted(4:j*15));
            media_Medida_3(j,1)=mean(Medida_3_converted(4:j*15));
            media_Medida_4(j,1)=mean(Medida_4_converted(4:j*15));

        else
            media_Medida_1(j,1)=mean(Medida_1_converted((15*j-14):j*15));
            media_Medida_2(j,1)=mean(Medida_2_converted((15*j-14):j*15));
            media_Medida_3(j,1)=mean(Medida_3_converted((15*j-14):j*15));
            media_Medida_4(j,1)=mean(Medida_4_converted((15*j-14):j*15));
        end
  end

Plot_orden = horzcat(media_Medida_1, media_Medida_2, media_Medida_3, media_Medida_4);

save  Basys28_vccaux_exp3.mat;
```

Anexo 10: Script Matlab. Experimento 4. Coeficiente de correlación.

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    C:\Users\yo\Desktop\ADC_PUF\Data_PUF\Data_PUF\Basys_20\Basys_20_T\Basys20_T_20mayo_1139_medida.txt
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

% Auto-generated by MATLAB on 2019/05/23 18:02:12

%% Initialize variables.
%filename = 'C:\Users\yo\Desktop\ADC_PUF\Data_PUF\Data_PUF\Basys_20\Basys_20_T\Basys20_T_20mayo_1139_medida.txt';
filename = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia1\TEST1\Arty_vccaux\medida';
filename1 = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia1\TEST1\Basys20_vccaux\medida';
filename2 = 'C:\Users\shado\OneDrive\Escritorio\Data_PUF\Dia1\TEST1\Basys28_vccaux\medida';
delimiter = {''};

%% Format for each line of text:
%   column1: text (%s)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');
fileID1 = fopen(filename1,'r');
fileID2 = fopen(filename2,'r');

%% Read columns of data according to the format.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);
dataArray1 = textscan(fileID1, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);
dataArray2 = textscan(fileID2, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'EmptyValue', NaN,  'ReturnOnError', false);

%% Close the text file.
fclose(fileID);
fclose(fileID1);
fclose(fileID2);

%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.


%% Create output variable
ArtyTempMedida = table(dataArray{1:end-1}, 'VariableNames', {'B0'});
Basys20Medida = table(dataArray1{1:end-1}, 'VariableNames', {'B1'});
Basys28Medida = table(dataArray2{1:end-1}, 'VariableNames', {'B2'});

%% Clear temporary variables
clearvars filename delimiter formatSpec fileID dataArray ans;
clearvars filename1 delimiter formatSpec fileID1 dataArray1 ans;
clearvars filename2 delimiter formatSpec fileID2 dataArray2 ans;

%%  Temperature conversion

% Arty_converted=hex2dec(ArtyTempMedida.B0)*503.975/4096-273.15;
% Basys20_converted=hex2dec(Basys20Medida.B1)*503.975/4096-273.15;
% Basys28_converted=hex2dec(Basys28Medida.B2)*503.975/4096-273.15;

%% Voltage conversion


Arty_converted=hex2dec(ArtyTempMedida.B0)*3/4096;
Basys20_converted=hex2dec(Basys20Medida.B1)*3/4096;
Basys28_converted=hex2dec(Basys28Medida.B2)*3/4096;

%% Mean data

  for j=1:448*9
      if j==1
          media_Arty(j,1)=mean(Arty_converted(4:j*15));
          media_Basys20(j,1)=mean(Basys20_converted(4:j*15));
          media_Basys28(j,1)=mean(Basys28_converted(4:j*15));
      else
          media_Arty(j,1)=mean(Arty_converted((15*j-14):j*15));
          media_Basys20(j,1)=mean(Basys20_converted((15*j-14):j*15));
          media_Basys28(j,1)=mean(Basys28_converted((15*j-14):j*15));
      end
  end
```

```
Plot_exp1 = horzcat(media_Arty,media_Basys20, media_Basys28);
C0 = corrcoef(media_Arty, media_Arty);
C1 = corrcoef(media_Arty, media_Basys20);
C2 = corrcoef(media_Arty, media_Basys28);
C3 = corrcoef(media_Basys20, media_Arty);
C4 = corrcoef(media_Basys20, media_Basys20);
C5 = corrcoef(media_Basys20, media_Basys28);
C6 = corrcoef(media_Basys28, media_Arty);
C7 = corrcoef(media_Basys28, media_Basys20);
C8 = corrcoef(media_Basys28, media_Basys28);

C ={'vccaux', 'Arty', 'Basys20', 'Basys28'; 'Arty', C0(1,2), C1(1,2) ,C2(1,2);'Basys20' ,C3(1,2), C4(1,2), C5(1,2);'Basys28', C6(1,2) ,C7(1,2) ,C8(1,2)};
save  vccaux_corr.mat;
```